Laura Lemay
Rafe Colburn

SIXTH EDITION

Covers HTML5

Sams Teach Yourself

Web Publishing with HTML and CSS

in One Hour a Day

SAMS

www.allitebooks.com

Laura Lemay
Rafe Colburn

Sams **Teach Yourself**

Web Publishing with HTML and CSS

in **One Hour** a Day

SAMS | 800 East 96th Street, Indianapolis, Indiana 46240

www.allitebooks.com

Sams Teach Yourself Web Publishing with HTML and CSS in One Hour a Day

Copyright © 2011 by Sams Publishing

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

ISBN-13: 978-0-672-33096-4

ISBN-10: 0-672-33096-2

Library of Congress Catalog-in-Publication data is on file.

First Printing: August 2010

Trademarks

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

Warning and Disclaimer

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

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearsoned.com

Acquisitions Editor
Mark Taber

Development Editor
Songlin Qiu

Managing Editor
Sandra Schroeder

Project Editor
Seth Kerney

Copy Editor
Keith Cline

Indexer
Brad Herriman

Proofreader
Apostrophe Editing Services, Inc.

Technical Editor
Julie Meloni

Publishing Coordinator
Vanessa Evans

Multimedia Developer
Dan Scherf

Book Designer
Gary Adair

Page Layout
Studio Galou, LLC

www.allitebooks.com

Contents at a Glance

PART I: Getting Started

1 Navigating the World Wide Web 7
2 Preparing to Publish on the Web 25
3 Introducing HTML and XHTML 49

PART II: Creating Web Pages

4 Learning the Basics of HTML 67
5 Organizing Information with Lists 81
6 Adding Links to Your Web Pages 99
7 Formatting Text with HTML and CSS 131

PART III: Doing More with HTML and CSS

8 Using CSS to Style a Site 173
9 Adding Images, Color, and Backgrounds 211
10 Building Tables 259
11 Designing Forms 311
12 Integrating Multimedia: Sound, Video, and More 355
13 Advanced CSS: Page Layout in CSS 389

PART IV: Using JavaScript and Ajax

14 Introducing JavaScript 411
15 Using JavaScript in Your Pages 435
16 Using JavaScript Libraries 459
17 Working with Frames and Linked Windows 489

PART V: Designing Effective Web Pages

18 Writing Good Web Pages: Do's and Don'ts 531
19 Designing for the Real World 567

PART VI: Going Live on the Web

20 Putting Your Site Online 593
21 Taking Advantage of the Server 619
22 Content Management Systems and Publishing Platforms 657

PART VII: Appendix

A Sources for Further Information 691

www.allitebooks.com

Table of Contents

Introduction 1

- Who Should Read This Book 2
- What This Book Contains 2
- What You Need Before You Start 3
- Conventions Used in This Book 4
  - Special Elements 4
  - HTML Input and Output Examples 5
  - Special Fonts 5
  - Workshop 5

PART I: Getting Started

LESSON 1: Navigating the World Wide Web 7

- How the World Wide Web Works 8
  - The Web Is a Hypertext Information System 8
  - The Web Is Graphical and Easy to Navigate 9
  - The Web Is Cross-Platform 10
  - The Web Is Distributed 10
  - The Web Is Dynamic 11
  - The Web Is Interactive 13
- Web Browsers 15
  - What the Browser Does 15
  - An Overview of Some Popular Browsers 16
  - Using the Browser to Access Other Services 18
- Web Servers 19
- Uniform Resource Locators 20
- Summary 21
- Workshop 21
  - Q&A 21
  - Quiz 22
  - Quiz Answers 22
  - Exercises 23

www.allitebooks.com

LESSON 2: Preparing to Publish on the Web 25

Anatomy of a Website 26
What Do You Want to Do on the Web? 28
Setting Your Goals 30
Breaking Up Your Content into Main Topics 31
Ideas for Organization and Navigation 32
Hierarchies 33
Linear 35
Linear with Alternatives 36
Combination of Linear and Hierarchical 38
Web 39
Storyboarding Your Website 42
What's Storyboarding and Why Do I Need It? 42
Hints for Storyboarding 43
Web Hosting 44
Using a Content-Management Application 44
Setting Up Your Own Web Hosting 45
Summary 46
Workshop 47
Q&A 47
Quiz 48
Quiz Answers 48
Exercises 48

LESSON 3: Introducing HTML and XHTML 49

What HTML Is (And What It Isn't) 50
HTML Describes the Structure of a Page 50
HTML Does Not Describe Page Layout 51
Why It Works This Way 52
How Markup Works 53
A Brief History of HTML Tags 53
The Current Standard: XHTML 1.1 54
The Future Standard: HTML5 55
What HTML Files Look Like 55
Text Formatting and HTML 60
Using Cascading Style Sheets 61
Including Styles in Tags 62

www.allitebooks.com

Programs to Help You Write HTML 62
Summary 64
Workshop 64
Q&A 64
Quiz 65
Quiz Answers 65
Exercises 65

PART II: Creating Web Pages

LESSON 4: Learning the Basics of HTML 67
Structuring Your HTML 68
The <html> Tag 68
The <head> Tag 69
The <body> Tag 69
The Title 70
Headings 72
Paragraphs 75
Comments 75
Summary 78
Workshop 78
Q&A 78
Quiz 79
Quiz Answers 79
Exercises 80

LESSON 5: Organizing Information with Lists 81
Lists: An Overview 82
Numbered Lists 83
Customizing Ordered Lists 84
Unordered Lists 87
Customizing Unordered Lists 88
Glossary Lists 90
Nesting Lists 92
Other Uses for Lists 94
Summary 95
Workshop 96

www.allitebooks.com

Q&A 96
Quiz 97
Quiz Answers 97
Exercises 98

LESSON 6: Adding Links to Your Web Pages 99

Creating Links 100
The Link Tag: <a> 100
Linking Local Pages Using Relative and Absolute Pathnames 105
Absolute Pathnames 106
Using Relative or Absolute Pathnames? 107
Links to Other Documents on the Web 108
Linking to Specific Places Within Documents 113
Creating Links and Anchors 113
Linking to Anchors in the Same Document 120
Anatomy of a URL 120
Parts of URLs 121
Special Characters in URLs 122
Additional Attributes for the <a> Tag 123
Kinds of URLs 123
HTTP 123
Anonymous FTP 124
Non-Anonymous FTP 125
Mailto 125
File 126
Summary 127
Workshop 128
Q&A 128
Quiz 130
Quiz Answers 130
Exercises 130

LESSON 7: Formatting Text with HTML and CSS 131

Character-Level Elements 132
Logical Styles 132
Physical Styles 135

www.allitebooks.com

Character Formatting Using CSS 137
The Text Decoration Property 137
Font Properties 138
Preformatted Text 139
Horizontal Rules 142
Attributes of the <hr> Tag 143
Line Break 145
Addresses 147
Quotations 147
Special Characters 149
Character Entities for Special Characters 150
Specifying Character Encoding 151
Character Entities for Reserved Characters 152
Text Alignment 153
Aligning Individual Elements 153
Aligning Blocks of Elements 153
Fonts and Font Sizes 155
Summary 167
Workshop 171
Q&A 171
Quiz 172
Quiz Answers 172
Exercises 172

PART III: Doing More with HTML and CSS

LESSON 8: Using CSS to Style a Site 173

Including Style Sheets in a Page 174
Creating Page-Level Styles 174
Creating Sitewide Style Sheets 175
Selectors 176
Contextual Selectors 176
Classes and IDs 177
What Cascading Means 179
Units of Measure 180
The Box Model 182
Borders 183

www.allitebooks.com

Margins and Padding 185
Controlling Size and Element Display 189
Float 192
CSS Positioning 196
Relative Positioning 197
Absolute Positioning 199
Controlling Stacking 202
The <body> Tag 205
Links 206
Summary 207
Workshop 207
Q&A 207
Quiz 208
Quiz Answers 208
Exercises 209

LESSON 9: Adding Images, Color, and Backgrounds 211

Images on the Web 212
Image Formats 213
GIF 213
JPEG 213
PNG 214
Inline Images in HTML: The <img> Tag 214
Adding Alternative Text to Images 215
Images and Text 219
Text and Image Alignment 220
Wrapping Text Next to Images 223
Adjusting the Space Around Images 226
Images and Links 228
Other Neat Tricks with Images 232
Image Dimensions and Scaling 232
More About Image Borders 233
Using Color 234
Specifying Colors 234
Changing Colors Using CSS 236
Changing Page Colors in HTML 236

www.allitebooks.com

Image Backgrounds 238
Specifying Backgrounds for Elements 241
Using Images As Bullets 242
What Is an Imagemap? 243
ImageMaps and Text-Only Browsers 244
Getting an Image 244
Determining Your Coordinates 245
The <map> and <area> Tags 248
The usemap Attribute 249
Image Etiquette 254
Summary 255
Workshop 256
Q&A 256
Quiz 257
Quiz Answers 258
Exercises 258

LESSON 10: Building Tables **259**

Creating Tables 260
Table Parts 260
The <table> Element 261
The Table Summary 261
Rows and Cells 262
Empty Cells 264
Captions 265
Sizing Tables, Borders, and Cells 269
Setting Table Widths 269
Changing Table Borders 270
Cell Padding 273
Cell Spacing 274
Column Widths 275
Setting Breaks in Text 278
Table and Cell Color 280
Aligning Your Table Content 282
Table Alignment 282
Cell Alignment 283
Caption Alignment 286

Spanning Multiple Rows or Columns 287
More Advanced Table Enhancements 296
Grouping and Aligning Columns 296
Grouping and Aligning Rows 300
Other Table Elements and Attributes 303
How Tables Are Used 303
Summary 304
Workshop 309
Q&A 309
Quiz 309
Quiz Answers 310
Exercises 310

LESSON 11: Designing Forms 311

Understanding Form and Function 312
Using the <form> Tag 317
Using the <label> Tag 320
Creating Form Controls with the <input> Tag 321
Creating Text Controls 322
Creating Password Controls 323
Creating Submit Buttons 324
Creating Reset Buttons 325
Creating Check Box Controls 325
Creating Radio Buttons 326
Using Images as Submit Buttons 327
Creating Generic Buttons 328
Hidden Form Fields 329
The File Upload Control 329
Using Other Form Controls 330
Using the button Element 330
Creating Large Text-Entry Fields with textarea 331
Creating Menus with select and option 332
Grouping Controls with fieldset and legend 340
Changing the Default Form Navigation 341
Using Access Keys 342
Creating disabled and readonly Controls 342

Applying Cascading Style Sheet Properties to Form Elements 343
Planning Your Forms 349
Summary 350
Workshop 351
Q&A 351
Quiz 352
Quiz Answers 352
Exercises 353

LESSON 12: Integrating Multimedia: Sound, Video, and More 355

Embedding Video the Simple Way 356
Advantages and Disadvantages of Hosting Videos on YouTube 357
Uploading Videos to YouTube 358
Customizing the Video Player 359
Other Services 360
Hosting Your Own Video 361
Video and Container Formats 362
Converting Video to H.264 363
Converting Video to Ogg Theora 366
Embedding Video Using <video> 366
The <video> Tag 367
Using the <source> Element 369
Embedding Flash Using the <object> Tag 370
Alternative Content for the <object> Tag 374
The <embed> Tag 375
Embedding Flash Movies Using SWFObject 376
Flash Video Players 378
JW Player 378
Using Flowplayer 380
Using the <object> Tag with the <video> Tag 382
Embedding Audio in Your Pages 383
The <audio> Tag 383
Flash Audio Players 384
Summary 385
Workshop 386
Q&A 386
Quiz 387

Quiz Answers 387
Exercises 388

LESSON 13: Advanced CSS: Page Layout in CSS 389

Laying Out the Page 390
The Problems with Layout Tables 390
Writing HTML with Structure 391
Writing a Layout Style Sheet 394
The Floated Columns Layout Technique 401
The Role of CSS in Web Design 403
Style Sheet Organization 404
Sitewide Style Sheets 407
Summary 408
Workshop 408
Q&A 408
Quiz 409
Quiz Answers 409
Exercises 410

PART IV: Using JavaScript and Ajax

LESSON 14: Introducing JavaScript 411

Why Would You Want to Use JavaScript? 412
Ease of Use 412
Increasing Server Efficiency 413
Integration with the Browser 413
The <script> Tag 414
The Structure of a JavaScript Script 414
The src Attribute 415
The JavaScript Language 415
Operators and Expressions 417
Variables 418
Control Structures 421
Functions 424
Data Types 426
Arrays 427
Objects 427

The JavaScript Environment 428
Events 429
Summary 432
Workshop 433
Q&A 433
Quiz 433
Quiz Answers 434
Exercises 434

LESSON 15: Using JavaScript in Your Pages 435

Validating Forms with JavaScript 436
Hiding and Showing Elements 443
Adding New Content to a Page 452
Summary 456
Workshop 456
Q&A 456
Quiz 457
Quiz Answers 457
Exercises 457

LESSON 16: Using JavaScript Libraries 459

What Are JavaScript Libraries? 460
Reviewing the Popular JavaScript Libraries 460
jQuery 461
Dojo 461
Yahoo! UI 461
Prototype 461
Other Libraries 462
Getting Started with jQuery 462
Your First jQuery Script 463
Binding Events 465
Modifying Styles on the Page 466
Hiding and Showing Elements 466
Retrieving and Changing Style Sheet Properties 467
Modifying Content on the Page 468
Manipulating Classes 468
Manipulating Form Values 471

Manipulating Attributes Directly 473
Adding and Removing Content 474
Special Effects 478
AJAX and jQuery 480
Using AJAX to Load External Data 481
Summary 485
Workshop 485
Q&A 485
Quiz 486
Quiz Answers 486
Exercises 487

LESSON 17: Working with Frames and Linked Windows 489

What Are Frames? 490
Working with Linked Windows 491
The <base> Tag 496
Working with Frames 498
The <frameset> Tag 499
The <frame> Tag 502
The <noframes> Tag 503
Changing Frame Borders 504
Creating Complex Framesets 507
Magic target Names 520
Inline Frames 521
Opening Linked Windows with JavaScript 523
Summary 526
Workshop 529
Q&A 529
Quiz 529
Quiz Answers 530
Exercises 530

PART V: Designing Effective Web Pages

LESSON 18: Writing Good Web Pages: Do's and Don'ts 531

Standards Compliance and Web Browsers 532
Progressive Enhancement 532

Choosing a Document Type 534
HTML5 535
Validating Your Pages 535
HTML Tidy 538
Writing for Online Publication 538
Write Clearly and Be Brief 539
Organize Your Pages for Quick Scanning 539
Make Each Page Stand on Its Own 541
Be Careful with Emphasis 541
Don't Use Browser-Specific Terminology 542
Spell Check and Proofread Your Pages 543
Design and Page Layout 543
Use Headings as Headings 543
Group Related Information Visually 544
Use a Consistent Layout 545
Using Links 546
Use Link Menus with Descriptive Text 546
Use Links in Text 547
Avoid the "Here" Syndrome 549
To Link or Not to Link 550
Using Images 552
Don't Overuse Images 552
Keep Images Small 553
Watch Out for Assumptions About Your Visitors' Hardware 555
Be Careful with Backgrounds and Link Colors 555
Making the Most of CSS and JavaScript 556
Put Your CSS and JavaScript in External Files 556
Location Matters 557
Shrink Your CSS and JavaScript 557
Other Good Habits and Hints 558
Link Back to Home 558
Don't Split Topics Across Pages 558
Don't Create Too Many or Too Few Pages 558
Sign Your Pages 561
Summary 562
Workshop 564

Q&A 564
Quiz 565
Quiz Answers 565
Exercises 566

LESSON 19: Designing for the Real World 567

What Is the Real World, Anyway? 568
Considering User Experience Level 569
Add a Search Engine 569
Use Concise, Sensible URLs 570
Navigation Provides Context 572
Are Your Users Tourists or Regulars? 573
Determining User Preference 573
Migrating to HTML5 575
Benefits of HTML5 575
What's Not in HTML5 577
Using HTML5 577
HTML5 and XML 578
XHTML 1.0 and HTML 4.01 579
What Is Accessibility? 579
Common Myths Regarding Accessibility 579
Section 508 580
Alternative Browsers 581
Writing Accessible HTML 582
Tables 582
Links 583
Images 584
Designing for Accessibility 586
Use Color 586
Fonts 586
Take Advantage of All HTML Tags 587
Frames and Linked Windows 587
Forms 587
Validating Your Sites for Accessibility 588
Summary 590
Workshop 590

Q&A 590
Quiz 591
Quiz Answers 592
Exercises 592

PART VI: Part VI: Going Live on the Web

LESSON 20: Putting Your Site Online 593

What Does a Web Server Do? 594
Other Things Web Servers Do 594
How to Find Web Hosting 595
Using a Web Server Provided by Your School or Work 595
Using a Commercial Web Host 596
Setting Up Your Own Server 597
Free Hosting 597
Organizing Your HTML Files for Publishing 598
Questions to Ask Your Webmaster 598
Keeping Your Files Organized with Directories 599
Having a Default Index File and Correct Filenames 599
Publishing Your Files 600
Moving Files Between Systems 600
Troubleshooting 603
I Can't Access the Server 603
I Can't Access Files 603
I Can't Access Images 604
My Links Don't Work 604
My Files Are Being Displayed Incorrectly 605
Registering and Advertising Your Web Pages 605
Getting Links from Other Sites 606
Promoting Your Site Through Social Media 606
Creating a Facebook Page for Your Site 607
Site Indexes and Search Engines 609
Google 609
Yahoo! 610
Microsoft Bing 610
Ask.com 610
Search Engine Optimization 611
Paying for Search Placement 612

Business Cards, Letterhead, Brochures, and Advertisements 612
Finding Out Who's Viewing Your Web Pages 612
Log Files 613
Google Analytics 614
Installing Google Analytics 614
Summary 616
Workshop 616
Q&A 617
Quiz 617
Quiz Answers 617
Exercises 618

LESSON 21: Taking Advantage of the Server 619

How PHP Works 620
Getting PHP to Run on Your Computer 621
The PHP Language 622
Comments 622
Variables 623
Arrays 624
Strings 626
Conditional Statements 628
PHP Conditional Operators 629
Loops 630
foreach Loops 630
for Loops 631
while and do...while Loops 632
Controlling Loop Execution 632
Built-in Functions 633
User-Defined Functions 634
Returning Values 635
Processing Forms 636
Handling Parameters with Multiple Values 637
Presenting the Form 642
Using PHP Includes 647
Choosing Which Include Function to Use 649
Expanding Your Knowledge of PHP 650

www.allitebooks.com

Database Connectivity 651
Regular Expressions 651
Sending Mail 652
Object-Oriented PHP 652
Cookies and Sessions 652
File Uploads 652
Other Application Platforms 652
Microsoft ASP.NET 653
Java EE 653
Ruby on Rails 653
Summary 653
Workshop 654
Q&A 654
Quiz 654
Quiz Answers 655
Exercises 655

LESSON 22: Content Management Systems and Publishing Platforms 657

The Rise of Content Management 658
Content Management in the Cloud 658
Is a Content Management System Right for You? 659
Types of Content Management Systems 660
Blogging Tools 660
Community Publishing Applications 661
Wikis 661
Image Galleries 662
General-Purpose Content Management Systems 663
Working with Packaged Software 664
Relational Databases 664
Deploying Applications 666
TypePad: A Hosted Blogging Application 667
WordPress 669
MediaWiki 674
Downloading and Installing MediaWiki 675
Using MediaWiki 676

Drupal 677
Using Drupal 678
Incorporating Dynamic Content from Other Sites into Your Pages 682
Using Photos from Flickr 682
Embedding Twitter Content 684
Integrating with Facebook 685
Other Applications 687
Spam 687
Summary 687
Workshop 688
Q&A 688
Quiz 689
Quiz Answers 689
Exercises 689

PART VI: Appendix

APPENDIX A: Sources for Further Information 691

Analytics 693
Browsers 693
Collections of HTML and Web Development Information 694
Imagemaps 695
HTML Editors and Converters 695
HTML Validators, Link Checkers, and Simple Spiders 695
JavaScript 696
Log File Parsers 696
HTML Style Guides 697
Servers and Server Administration 697
Sound and Video 697
Specifications for HTML, HTTP, and URLs 698
Server-Side Scripting 698
Web Publishing Tools 699
Other Web-Related Topics 699
Tools and Information for Images 699
Web Hosting Providers 700

INDEX 701

About the Authors

Rafe Colburn is an author and web developer with more than 15 years experience building websites. His other books include *Special Edition Using SQL* and *Sams Teach Yourself CGI in 24 Hours*. You can read his blog at http://rc3.org or find him on Twitter as@rafeco.

Laura Lemay is one of the world's most popular authors on web development topics. She is the author of *Sams Teach Yourself Web Publishing with HTML*, *Sams Teach Yourself Java in 21 Days*, and *Sams Teach Yourself Perl in 21 Days*.

Dedication

For Patricia.

Acknowledgments

I'd like to acknowledge the hard work of all the people at Sams Publishing who clean up my messes and get these books out on the shelves. Special thanks go to Mark Taber, Songlin Qiu, Seth Kerney, and technical editor Julie Meloni. I'd also like to thank my wife for suffering through yet another one of these projects.

—Rafe Colburn

We Want to Hear from You!

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

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

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

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

Email: feedback@samspublishing.com

Mail: Mark Taber
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Over the past decade, the Web has become completely integrated into the fabric of society. Most businesses have websites, and it's rare to see a commercial on television that doesn't display a URL. The simple fact that most people now know what a URL *is* speaks volumes. People who didn't know what the Internet was several years ago are now reconnecting with their high school friends on Facebook.

Perhaps the greatest thing about the Web is that you don't have to be a big company to publish things on it. The only things you need to create your own website are a computer with access to the Internet and the willingness to learn. Obviously, the reason you're reading this is that you have an interest in web publishing. Perhaps you need to learn about it for work, or you're looking for a new means of self-expression, or you want to post baby pictures on the Web so that your relatives all over the country can stay up-to-date. The question is, how do you get started?

There's more than enough information on the Web about how to publish websites like a seasoned professional. There are tutorials, reference sites, tons of examples, and free tools to make it easier to publish on the Web. However, the advantage of reading this book instead is that all the information you need to build websites is organized in one place and presented in an orderly fashion. It has everything you need to master HTML, publish sites to a server on the Web, create graphics for use on the Web, and keep your sites running smoothly.

But wait, there's more. Other books on how to create web pages just teach you the basic technical details, such as how to produce a boldface word. In this book, you'll also learn why you should be producing a particular effect and when you should use it. In addition, this book provides hints, suggestions, and examples of how to structure your overall website, not just the words on each page. This book won't just teach you how to create a website—it'll teach you how to create a good website.

Right now, the Web is transitioning from HTML 4/XHTML 1, the standards on which browsers have been based since the late nineties, to HTML5, a new standard that, at the time of publishing, has not been finalized. Browsers have already started adding support for HTML5, and after Internet Explorer 9 is released, all the major browsers will offer substantial support for it. In this book, examples are written in valid HTML5 using tags that also work in all current browsers wherever possible. Exceptions and caveats are noted whenever I use tags that are not valid in HTML5 or were not part of HTML 4.

Who Should Read This Book

Is this book for you? That depends:

- If you've seen what's out on the Web and you want to contribute your own content, this book is for you.
- If you work for a company that wants to create a website and you're not sure where to start, this book is for you.
- If you're an information developer, such as a technical writer, and you want to learn how the Web can help you present your information online, this book is for you.
- If you're just curious about how the Web works, some parts of this book are for you, although you might be able to find what you need on the Web itself.
- If you've created web pages before with text, images, and links, and you've played with a table or two and set up a few simple forms, you may be able to skim the first half of the book. The second half should still offer you a lot of helpful information.

If you've never seen the Web before but you've heard that it's really nifty, this book isn't for you. You'll need a more general book about the Web before you can produce websites yourself.

What This Book Contains

The lessons are arranged in a logical order, taking you from the simplest tasks to more advanced techniques:

- **Part I: Getting Started**

  In Part I, you'll get a general overview of the World Wide Web and what you can do with it, and then you'll come up with a plan for your web presentation. You'll also write your first (very basic) web page.

- **Part II: Creating Web Pages**

  In Part II, you'll learn how to write simple documents in the HTML language and link them together using hypertext links. You'll also learn how to format your web pages and how to use images on your pages.

- **Part III: Doing More with HTML and CSS**

  In Part III, you'll learn how to create tables and forms and place them on your pages. You'll also learn how to use cascading style sheets to describe how your pages are formatted instead of tags that are focused strictly on formatting.

- **Part IV: Using JavaScript and AJAX**

  In Part IV, we'll look at how you can extend the functionality of your web pages by adding JavaScript to them. First, we provide an overview of JavaScript, and then we provide some specific JavaScript examples you can use on your own pages. Finally, we describe how you can dynamically modify the look and feel of your pages using Dynamic HTML.

- **Part V: Designing Effective Web Pages**

  Part V gives you some hints for creating a well-constructed website, and you'll explore some sample websites to get an idea of what sort of work you can do. You'll learn how to design pages that will reach the types of real-world users you want to reach, and you'll learn how to create an accessible site that is usable by people with disabilities.

- **Part VI: Going Live on the Web**

  In Part VI, you'll learn how to put your site up on the Web, including how to advertise the work you've done. You'll also learn how to use some of the features of your web server to make your life easier.

What You Need Before You Start

There are lots of books about how to use the Web. This book isn't one of them. We're assuming that if you're reading this book, you already have a working connection to the Internet, you have a web browser such as Microsoft Internet Explorer, Mozilla Firefox, Apple's Safari, or Google's Chrome, and you're familiar with the basics of how the Web and the Internet work. You should also have at least a passing acquaintance with some other elements of the Internet, such as email and FTP, because we refer to them in general terms in this book.

In other words, you need to have used the Web to provide content for the Web. If you meet this one simple qualification, read on!

NOTE

To really take advantage of all the concepts and examples in this book, you should consider using the most recent version of Microsoft Internet Explorer (version 8.0 or later), Mozilla Firefox (version 3.0 or later), Safari (version 4 or later), or Google Chrome.

Conventions Used in This Book

This book uses special typefaces and other graphical elements to highlight different types of information.

Special Elements

Three types of "boxed" elements present pertinent information that relates to the topic being discussed: Note, Tip, and Caution as follows:

NOTE

Notes highlight special details about the current topic.

TIP

It's a good idea to read the tips because they present shortcuts or trouble-saving ideas for performing specific tasks.

CAUTION

Don't skip the cautions. They help you avoid making bad decisions or performing actions that can cause you trouble.

▼ Task

▲ Tasks demonstrate how you can put the information in a lesson into practice by giving you a real working example.

HTML Input and Output Examples

Throughout the book, we present exercises and examples of HTML input and output.

Input ▼

An input icon identifies HTML code that you can type in yourself.

Output ▼

An output icon indicates the results of the HTML input in a web browser such as Microsoft Internet Explorer.

Special Fonts

Several items are presented in a monospace font, which can be plain or italic. Here's what each one means:

`plain mono`—Applied to commands, filenames, file extensions, directory names, Internet addresses, URLs, and HTML input. For example, HTML tags such as `<TABLE>` and `<P>` appear in this font.

`mono italic`—Applied to placeholders. A placeholder is a generic item that replaces something specific as part of a command or computer output. For instance, the term represented by *`filename`* would be the real name of the file, such as *`myfile.txt`*.

Workshop

In the "Workshop" section, you can reinforce your knowledge of the concepts in the lesson by answering quiz questions or working on exercises. The Q&A provides additional information that didn't fit in neatly elsewhere in the lesson.

www.allitebooks.com

This page intentionally left blank

LESSON 1
Navigating the World Wide Web

A journey of a thousand miles begins with a single step, and here you are in Lesson 1 of a journey that will show you how to write, design, and publish pages on the World Wide Web. But before beginning the actual journey, you should start simple, with the basics. You'll learn the following:

- How the World Wide Web really works
- What web browsers do, and a couple of popular ones from which to choose
- What a web server is, and why you need one
- Some information about *uniform resource locators (URLs)*

These days, the Web is pervasive, and maybe most if not all of today's information will seem like old news. If so, feel free to skim this lesson and skip ahead to Lesson 2, "Preparing to Publish on the Web," where you'll find an overview of points to think about when you design and organize your own Web documents.

How the World Wide Web Works

Chances are that you've used the Web, perhaps even a lot. However, you might not have done a lot of thinking about how it works under the covers. In this first section, I describe the Web at a more theoretical level so that you can understand how it works as a platform.

I have a friend who likes to describe things using many meaningful words strung together in a chain so that it takes several minutes to sort out what he's just said.

If I were he, I'd describe the World Wide Web as a global, interactive, dynamic, cross-platform, distributed, graphical hypertext information system that runs over the Internet. Whew! Unless you understand all these words and how they fit together, this description isn't going to make much sense. (My friend often doesn't make much sense, either.)

So, let's look at all these words and see what they mean in the context of how you use the Web as a publishing medium.

The Web Is a Hypertext Information System

The idea behind hypertext is that instead of reading text in a rigid, linear structure (such as a book), you can skip easily from one point to another. You can get more information, go back, jump to other topics, and navigate through the text based on what interests you at the time.

Hypertext enables you to read and navigate text and visual information in a nonlinear way, based on what you want to know next.

When you hear the term *hypertext*, think *links*. (In fact, some people still refer to links as hyperlinks.) Whenever you visit a web page, you're almost certain to see links throughout the page. Some of the links might point to locations within that same page, others to pages on the same site, and still others might point to pages on other sites. Hypertext was an old concept when the Web was invented—it was found in applications such as HyperCard and various help systems. However, the World Wide Web redefined how large a hypertext system could be. Even large websites were hypertext systems of a scale not before seen, and when you take into account that it's no more difficult to link to a document on a server in Australia from a server in the United States than it is to link to a document stored in the same directory, the scope of the Web becomes truly staggering.

NOTE

Nearly all large corporations and medium-sized businesses and organizations are using web technology to manage projects, order materials, and distribute company information in a paperless environment. By locating their documents on a private, secure web server called an *intranet*, they take advantage of the technologies the World Wide Web has to offer while keeping the information contained within the company.

The Web Is Graphical and Easy to Navigate

In the early days, using the Internet involved simple text-only applications. You had to navigate the Internet's various services using command-line programs (think DOS) and arcane tools. Although plenty of information was available on the Net, it wasn't necessarily pretty to look at or easy to find.

Then along came the first graphical web browser: Mosaic. It paved the way for the Web to display both text and graphics in full color on the same page. The ability to create complex, attractive pages rivaling those founds in books, magazines, and newspapers propelled the popularity of the Web. These days, the Web offers such a wide degree of capabilities that people are writing web applications that replace desktop applications.

A *browser* is used to view and navigate web pages and other information on the World Wide Web. Currently, the most popular browsers are Microsoft Internet Explorer, Mozilla Firefox, Apple Safari, and Google Chrome. In addition, more and more people are using mobile devices to access the Web, most of which have their own browsers.

Hypertext or Hypermedia?

If the Web incorporates so much more than text, why do I keep calling the Web a hypertext system? Well, if you're going to be absolutely technically correct about it, the Web is not a hypertext system—it's a hyper*media* system. But, on the other hand, you might argue that the Web began as a text-only system, and much of the content is still text-heavy, with extra bits of media added in as emphasis. I prefer the term *hypertext*, and it's my book, so I use it. You know what I mean.

The Web Is Cross-Platform

If you can access the Internet, you can access the World Wide Web, regardless of whether you're working on a low-end PC or a fancy expensive workstation. These days, you can even access the Web from most mobile phones. If you think Windows menus and buttons look better than Macintosh menus and buttons or vice versa (or if you think both Macintosh and Windows people are weenies), it doesn't matter. The World Wide Web isn't limited to any one kind of machine or developed by any one company. The Web is entirely cross-platform.

Cross-platform means that you can access web information equally well from any computer hardware running any operating system using any display.

The Cross-Platform Ideal

The whole idea that the Web is—and should be—cross-platform is strongly held to by purists. The reality, however, is somewhat different. With the introduction over the years of numerous special features, technologies, and media types, the Web has lost some of its capability to be truly cross-platform. As web authors choose to use these nonstandard features, they willingly limit the potential audience for the content of their sites. For example, a site centered on a Flash animation is essentially unusable for someone using a browser that doesn't have a Flash player, or for a user who might have turned off Flash for quicker downloads. Similarly, some programs that extend the capabilities of a browser (known as *plug-ins*) are available only for one platform (either Windows, Macintosh, or UNIX). Choosing to use one of those plug-ins makes that portion of your site unavailable to users who are either on the wrong platform or don't want to bother to download and install the plug-in.

The Web Is Distributed

Web content can take up a great deal of space, particularly when you include images, audio, and video. To store all the information, graphics, and multimedia published on the Web, you would need an untold amount of disk space, and managing it would be almost impossible. (Not that there aren't people who try.) Imagine that you were interested in finding out more information about alpacas (Peruvian mammals known for their wool), but when you selected a link in your online encyclopedia, your computer prompted you to insert CD-ROM #456 ALP through ALR. You could be there for a long time just looking for the right CD-ROM!

The Web succeeds at providing so much information because that information is distributed globally across millions of websites, each of which contributes the space for the information it publishes. These sites reside on one or more computers, referred to as web servers. A *web server* is just a computer that listens for requests from web browsers and responds to that request. You, as a consumer of that information, request a resource from the server to view it. You don't have to install it, change disks, or do anything other than point your browser at that site.

A *website* is a location on the Web that publishes some kind of information. When you view a web page, your browser connects to that website to get that information.

Each website, and each page or bit of information on that site, has a unique address. This address is called a *uniform resource locator* or URL. When people tell you to visit a site at http://www.yahoo.com/, they've just given you a URL. Whenever you use a browser to visit a website, you get there using a URL. You'll learn more about URLs later in this lesson in the "Uniform Resource Locators" section.

The Web Is Dynamic

If you want a permanent copy of some information that's stored on the Web, you have to save it locally because the content can change any time, even while you're viewing the page.

If you're browsing that information, you don't have to install a new version of the help system, buy another book, or call technical support to get updated information. Just launch your browser and check out what's there.

If you're publishing on the Web, you can make sure that your information is up-to-date all the time. You don't have to spend a lot of time re-releasing updated documents. There's no cost of materials. You don't have to get bids on numbers of copies or quality of output. Color is free. And you won't get calls from hapless customers who have a version of the book that was obsolete 4 years ago.

Consider a book published and distributed entirely online, such as *Little Brother* by Cory Doctorow (which you can find at http://craphound.com/littlebrother/). He can correct any mistakes in the book and simply upload the revised text to his website, making it instantly available to his readers. He can post pointers to foreign language translations of the book as they arrive. The website for the book appears in Figure 1.1.

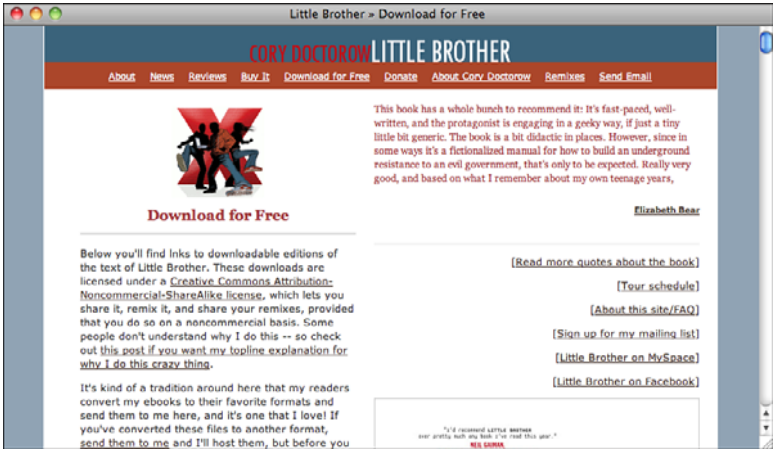


FIGURE 1.1
The website for *Little Brother*.

NOTE

The pictures throughout this book usually are taken in Safari running on Mac OS X. The only reason for this use is that I'm writing this book on an Apple Macintosh. If you're using a different operating system, don't feel left out. As I noted earlier, the glory of the Web is that you see the same information regardless of the platform you use.

For some sites, the capability to update the site on-the-fly, at any moment, is precisely why the site exists. Figure 1.2 shows the home page for Yahoo! News, an online news site that's updated 24 hours a day to reflect up-to-the-minute news as it happens. Because the site is up and available all the time, it has an immediacy that neither hard-copy newspapers nor most television news programs can match. Visit Yahoo! News at http://news.yahoo.com.

These days, you don't even need to reload a web page to receive updated information. Through the use of JavaScript, which I discuss starting in Lesson 14, "Introducing JavaScript," you can update the contents of a page in real time. The scores and statistics on the NBA game page in Figure 1.3 are updated in place as the game progresses.

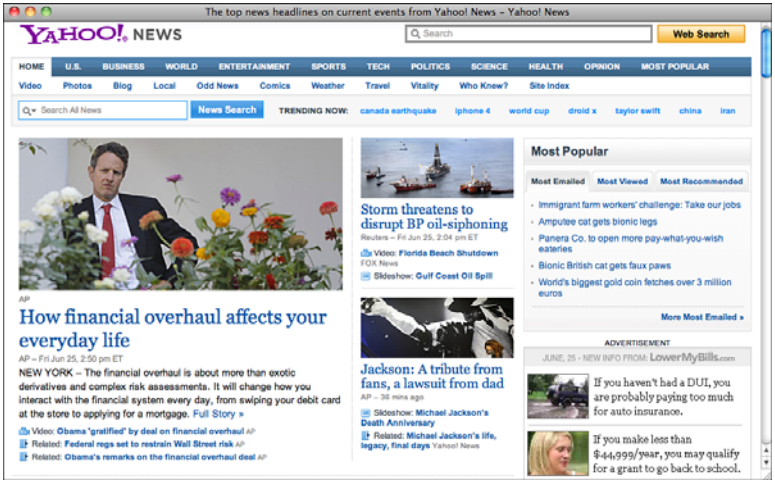


FIGURE 1.2
Yahoo! News.

FIGURE 1.3
Live game updates on the CBS Sports website.

The Web Is Interactive

Interactivity is the capability to "talk back" to the web server. More traditional media, such as television, isn't interactive in the slightest; all you do is sit and watch as shows are played at you. Other than changing the channel, you don't have much control over what you see. The Web is inherently interactive; the act of selecting a link and jumping to another web page to go somewhere else on the Web is a form of interactivity. In addition to this simple interactivity, however, the Web enables you to communicate with the publisher of the pages you're reading and with other readers of those pages.

For example, pages can be designed to contain interactive forms that readers can fill out. Forms can contain text-entry areas, radio buttons, or simple menus of items. When the form is submitted, the information typed by readers is sent back to the server from which the pages originated. Figure 1.4 shows an example of an online form.

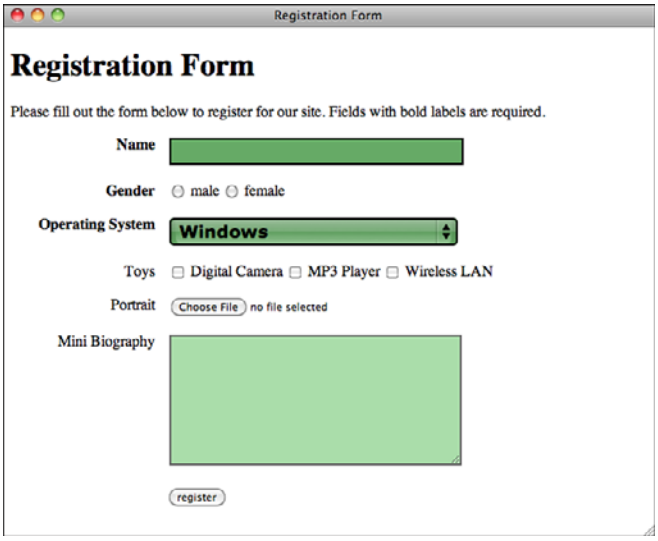


FIGURE 1.4
A registration form.

As a publisher of information on the Web, you can use forms for many different purposes, such as the following:

- To get feedback about your pages.
- To get information from your readers (survey, voting, demographic, or any other kind of data). You then can collect statistics on that data, store it in a database, or do anything you want with it.
- To provide online order forms for products or services available on the Web.
- To create comment forms and forums that enable your readers to post their own information on your pages. These kinds of systems enable your readers to communicate not only with you, but also with other readers of your pages.

In addition to forms, which provide some of the most popular forms of interactivity on the Web, advanced features of web technologies provide even more interactivity. Flash, and JavaScript, for example, enable you to include entire programs and games inside web pages. Software can run on the Web to enable real-time chat sessions between your readers. As time goes on, the Web becomes less of a medium for people passively sitting and digesting information (and becoming "Net potatoes") and more of a medium for reaching and communicating with other people all over the world.

Web Browsers

A web browser, as mentioned earlier, is the application you use to view pages and navigate the World Wide Web. A wide array of Web browsers is available for just about every platform you can imagine. Microsoft Internet Explorer, for example, is included with Windows, and Safari is included with Mac OS X. Mozilla Firefox, Google Chrome, and Opera are all available as free downloads. Currently, the most widely used is Microsoft Internet Explorer (sometimes called just *Internet Explorer* or *IE*), but competing browsers are increasing their share of the market. These days, if you don't take all the popular browsers into account when creating your Web pages, you'll limit your audience substantially.

NOTE

Choosing to develop for a specific browser, such as Internet Explorer, is suitable when you know a limited audience using the targeted browser software will view your website. Developing this way is a common practice in corporations implementing intranets. In these situations, it's a fair assumption that all users in the organization will use the browser supplied to them and, accordingly, it's possible to design the web pages on an intranet to use the specific capabilities of the browser in question.

What the Browser Does

The core purpose of a web browser is to connect to web servers, request documents, and then properly format and display those documents. Web browsers can also display files on your local computer, download files that are not meant to be displayed, and in some cases even allow you to send and retrieve email. What the browser is best at, however, is dealing with retrieving and displaying web documents. Each web page is a file written in a language called the *Hypertext Markup Language (HTML)* that includes the text of the page, a description of its structure, and links to other documents, images, or other media. The browser takes the information it gets from the web server and formats it for your system. Different browsers might format and display the same file in diverse ways, depending on the capabilities of that system and how the browser is configured.

Retrieving documents from the Web and formatting them for your system are the two tasks that make up the core of a browser's functionality. Depending on the browser you use and the features it includes, however, you can also play Flash animations, play multimedia files, run Java applets, read your mail, or use other advanced features that a particular browser offers.

www.allitebooks.com

An Overview of Some Popular Browsers

This section describes the most popular browsers currently on the Web. They're in no way the only browsers available, and if the browser you're using isn't listed here, don't feel that you have to use one of these. Whichever browser you have is fine as long as it works for you.

Microsoft Internet Explorer

Microsoft's browser, Microsoft Internet Explorer, is included with Microsoft Windows. Because Windows has the largest market share, Internet Explorer is the most popular web browser. However, many users choose to replace Internet Explorer with other browsers because of security concerns and greater support for web standards and newer capabilities.

NOTE

If you're serious about web design, you should install all the popular browsers on your system and use them to view your pages after you've published them. That way, you can make sure that everything is working properly. Even if you don't use a particular browser on a day-to-day basis, your site will be visited by people who do. If you are interested in checking cross-browser compatibility issues, start with Microsoft Internet Explorer and Mozilla Firefox, and include Google Chrome, too.

The percentage of users who use Internet Explorer varies widely from site to site. Current estimates are that various versions of Internet Explorer comprise about 55% of the browser market, but the browser usage varies widely from site to site. Figure 1.5 shows Internet Explorer running under Windows 7.

One other important point to make about Internet Explorer is that the different versions of IE differ greatly. Version 8 of Internet Explorer was released in 2009, but many users haven't upgraded from IE 7 or even IE 6. IE differs widely between versions, so to get a site to work properly, you need to test in each version. Web publishers are just starting to drop support for IE 6, and Microsoft recommends that all users upgrade to a newer version.

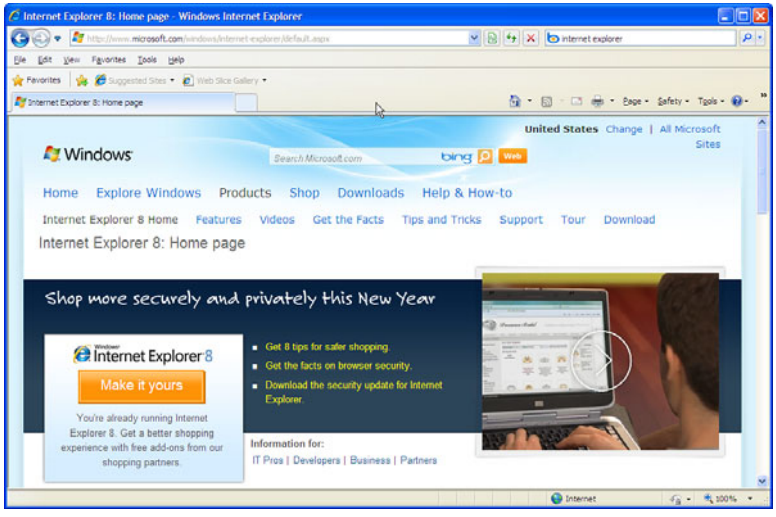


FIGURE 1.5
Microsoft Internet Explorer (Windows 7).

Mozilla Firefox

Mozilla Firefox is a free, open source web browser that enjoys more than 30% of the browser market as of June, 2010. Netscape Navigator was the first popular commercial web browser. Version 1.0 was released in 1994. In 1998, Netscape Communications opened the source code to its web browser and assigned some staff members to work on making it better. Seven years and many releases later, the result of that effort was Mozilla Firefox. Netscape Communications, since acquired by America Online, no longer has any official ties to the Mozilla Foundation, which is now an independent nonprofit organization.

Firefox has become popular in large part because it is free from the security issues that plague Internet Explorer. In addition, a large number of Firefox extensions improve the browser experience, and Firefox has done a good job of keeping up with web standards as they have evolved. Firefox is available for Windows, Mac OS X, and Linux and is a free download at http://www.mozilla.com.

I recommend that you use Firefox as you work through this book, for two reasons. The first is that there are a number of Firefox add-ons that will make your life much easier when you develop websites. I discuss them in Lesson 2. The second is that getting your pages to work in Firefox is a good place to start before you test them in other browsers. Generally speaking, it's easier to make a page that works in Firefox work in Internet Explorer than it is to make a page that works in Internet Explorer work in Firefox (along with the other browsers).

Apple Safari

Safari is the default browser for OS X. There is also a version that's available for Windows, and a mobile version of this browser is installed on the Apple iPhone. It is based on open source technology, and its support for web standards is at a similar level to Firefox. Right now, Safari has about 5% of the browser market.

Google Chrome

Google Chrome is the new kid on the block. It uses the same HTML engine as Safari, an open source engine called WebKit. Google Chrome is known for offering very high performance, and has some features that prevent it from crashing as often as other browsers. It's a free download and despite that it was released at the end of 2008, Chrome users make up about 5% of the total.

Other Browsers

When it comes to browsers, Microsoft Internet Explorer and Mozilla Firefox are the big two. And in terms of market share, Internet Explorer has the majority, but plenty of other browsers are floating around, too. You'd think that given that the browser market has been dominated by Microsoft or Netscape almost since its inception, there wouldn't be a lot of other browsers out there, but that's not the case.

For example, Opera (http://www.operasoftware.com/) has a niche market. It's small, fast, free, and available for a number of platforms, including Windows, Mac OS X, and Linux. It's also standards-compliant. For UNIX users who use KDE, there's Konqueror. There are various Mozilla offshoots, such as Camino for Mac OS X. Likewise, command-line browsers such as Lynx and Links are available to provide an all-text view of web pages. There are also a number of browsers that provide access to the Web for people with various special needs; I discuss them in detail in Lesson 19, "Designing for the Real World." It makes sense to code to common standards to accommodate all these types of browsers.

Using the Browser to Access Other Services

Internet veterans know that there are dozens of different ways to get information: FTP, Usenet news, and email. Before the Web became as popular as it is now, you had to use a different tool for each of these, all of which used different commands. Although all these choices made for a great market for *How to Use the Internet* books, they weren't easy to use.

Web browsers changed that. Although the Web is its own information system with its own Internet protocol (the *Hypertext Transfer Protocol* or *HTTP*), web browsers can read files from other Internet services also. Even better, you can create links to information on those systems just as you would create links to web pages. This process is seamless and available through a single application.

To point your browser to different kinds of information on the Internet, you use different kinds of URLs. Most URLs start with http:, which indicates a file at an actual website. To download a file from a public site using FTP, you'd use a URL like ftp://_name_of_site/directory/filename. You can also view the contents of a directory on a publicly accessible FTP site using an ftp: URL that ends with a directory name. Figure 1.6 shows a listing of files from the iBiblio FTP site at ftp://ftp.ibiblio.org/.

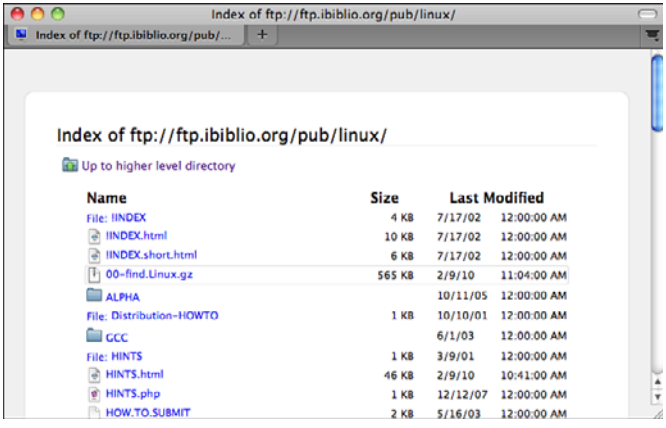


FIGURE 1.6
A listing of files and directories available at the iBiblio FTP site.

To access a Usenet newsgroup through your web browser (thereby launching an external news-reading program), you can simply enter a news: URL, such as news:alt.usage.english.

You'll learn more about different kinds of URLs in Lesson 6, "Adding Links to Your Web Pages."

Web Servers

To view and browse pages on the Web, all you need is a web browser. To publish pages on the Web, you need a web server.

A *web server* is the program that runs on a computer and is responsible for replying to web browser requests for files. You need a web server to publish documents on the Web. One point of confusion is that the computer on which a server program runs is also referred to as a server. So, when someone uses the term *web server*, she could be referring to a program used to distribute web pages or the computer on which that program runs.

When you use a browser to request a page on a website, that browser makes a web connection to a server using HTTP. The server accepts the connection, sends the contents of the requested files, and then closes the connection. The browser then formats the information it got from the server.

On the server side, many different browsers can connect to the same server to get the same information. The web server is responsible for handling all these requests.

Web servers do more than just serve files. They're also responsible for managing form input and for linking forms and browsers with programs such as databases running on the server.

As with browsers, many different servers are available for many different platforms, each with many different features. For now, all you need to know is what the server is there for; you'll learn more about web servers in Lesson 20, "Putting Your Site Online."

These days, a lot of people make websites without uploading pages to a web server. They publish blogs using any of a number of popular services, or they use a content management system of some kind, or they publish pages on a wiki. Even using Twitter and posting status updates on Facebook are forms of web publishing. Regardless of the application you use to publish information on the Web, it is likely to be published as HTML, and understanding how HTML works can help you achieve the results you desire.

Uniform Resource Locators

As you learned earlier, a URL is a pointer to some bit of data on the Web, be it a web document, a file available via FTP, a posting on Usenet, or an email address. The URL provides a universal, consistent method for finding and accessing information.

In addition to typing URLs directly into your browser to go to a particular page, you also use URLs when you create a hypertext link within a document to another document. So, any way you look at it, URLs are important to how you and your browser get around on the Web.

URLs contain information about the following:

- How to get to the information (which protocol to use: FTP, HTTP, or file)
- The Internet hostname of the computer where the content is stored (www.ncsa.uiuc.edu, ftp.apple.com, netcom16.netcom.com, and so on)
- The directory or other location on that site where the content is located

You also can use special URLs for tasks such as sending mail to people (called *Mailto URLs*) and running JavaScript code. You'll learn all about URLs and what each part means in Lesson 6.

Summary

To publish on the Web, you have to understand the basic concepts that make up the parts of the Web. In this lesson, you learned three major concepts. First, you learned about a few of the more useful features of the Web for publishing information. Second, you learned about web browsers and servers and how they interact to deliver web pages. Third, you learned about what a URL is and why it's important to web browsing and publishing.

Workshop

Each lesson in this book contains a workshop to help you review the topics you learned. The first section of this workshop lists some common questions about the Web. Next, you'll answer some questions that I'll ask you about the Web. The answers to the quiz appear in the next section. At the end of each lesson, you'll find some exercises that can help you retain the information you learned about the Web.

Q&A

Q Who runs the Web? Who controls all these protocols? Who's in charge of all this?

A No single entity owns or controls the World Wide Web. Given the enormous number of independent sites that supply information to the Web, for any single organization to set rules or guidelines would be impossible. Two groups of organizations, however, have a great influence over the look and feel and direction of the Web itself.

The first is the World Wide Web Consortium (W3C), based at Massachusetts Institute of Technology in the United States and INRIA in Europe. The W3C is

made up of individuals and organizations interested in supporting and defining the languages and protocols that make up the Web (HTTP, HTML, XHTML, and so on). It also provides products (browsers, servers, and so on) that are freely available to anyone who wants to use them. The W3 Consortium is the closest anyone gets to setting the standards for and enforcing rules about the World Wide Web. You can visit the Consortium's home page at http://www.w3.org/.

The second group of organizations that influences the Web is the browser developers themselves, most notably Microsoft and the Mozilla Foundation. The competition to be the most popular and technically advanced browser on the Web can be fierce. Although both organizations claim to support and adhere to the guidelines proposed by the W3C, both also include their own new features in new versions of their software—features that sometimes conflict with each other and with the work the W3C is doing.

Things still change pretty rapidly on the Web. The popular browsers are finally converging to support many of the standards defined by the W3C, so writing to those standards will work most of the time. I talk about the exceptions throughout this book.

Q I've heard that the Web changes so fast that it's almost impossible to stay current. Is this book doomed to be out-of-date the day it's published?

A Although it's true that things do change on the Web, the vast majority of the information in this book can serve you well far into the future. HTML is as stable now as it has ever been, and when you learn the core technologies of *Hypertext Markup Language (HTML)*, *Cascading Style Sheets (CSS)*, and JavaScript, you can add on other things at your leisure.

Quiz

1. What's a URL?
2. What's required to publish documents on the Web?

Quiz Answers

1. A URL, or uniform resource locator, is an address that points to a specific document or bit of information on the Internet.
2. You need access to a web server. Web servers, which are programs that serve up documents over the Web, reply to web browser requests for files and send the requested pages to many different types of browsers. They also manage form input and handle database integration.

Exercises

1. Try navigating to each of the different types of URLs mentioned in this lesson (http:, ftp:, and news:). Some links you might want to try are http://www.tywebpub.com and ftp://ftp.ibiblio.org.
2. Download a different browser than the one you ordinarily use and try it out for a while. If you're using Internet Explorer, try out Firefox, Chrome, Safari, or even a command-line browser such as Lynx or Links. To see how things have changed and how some users who don't upgrade their browser experience the Web, download an old browser from http://browsers.evolt.org/ and try it out.

This page intentionally left blank

LESSON 2
Preparing to Publish on the Web

When you write a book, a paper, an article, or even a memo, you usually don't just jump right in with the first sentence and then write it through to the end. The same goes with the visual arts—you don't normally start from the top-left corner of the canvas or page and work your way down to the bottom right.

A better way to write, draw, or design a work is to do some planning beforehand—to know what you're going to do and what you're trying to accomplish, and to have a general idea or rough sketch of the structure of the piece before you jump in and work on it.

Just as with more traditional modes of communication, the process of writing and designing web pages takes some planning and thought before you start flinging text and graphics around and linking them wildly to each other. It's perhaps even more important to plan ahead with web pages because trying to apply the rules of traditional writing or design to online hypertext often results in documents that are either difficult to understand and navigate online or that simply don't take advantage of the features that hypertext provides. Poorly organized web pages also are difficult to revise or to expand.

The other question you'll want to ask is where your website will be hosted. In this lesson, I explain how to make provisions for putting your site on the Web. I describe some of the things you should think about before you begin developing your web pages. Specifically, you need to do the following:

- Learn the differences between a web server, a website, a web page, and a home page.
- Think about the sort of information (content) you want to put on the Web.

www.allitebooks.com

- Set the goals for the website.
- Organize your content into the main topics.
- Come up with a general structure for pages and topics.
- Use storyboarding to plan your website.

Anatomy of a Website

First, here's a look at some simple terminology I use throughout this book. You need to know what the following terms mean and how they apply to the body of work you're developing for the Web:

- **Website**—A collection of one or more web pages linked together in a meaningful way that, as a whole, describes a body of information or creates an overall effect (see Figure 2.1).
- **Web server**—A computer on the Internet or an intranet that delivers web pages and other files in response to browser requests. (An intranet is a network that uses Internet protocols but is not publicly accessible.)
- **Web page**—A single document on a website, usually consisting of an *Hypertext Markup Language (HTML)* document and any items that are displayed within that document, such as inline images.
- **Home page**—The entry page for a website, which can link to additional pages on the same website or pages on other sites.

Each website is stored on a web server. Throughout the first few lessons in this book, you learn how to develop well thought-out and well-designed websites. Later, you learn how to publish your site on an actual web server.

A *web page* is an individual element of a website in the same way that a page is a single element of a book or a newspaper. (Although, unlike paper pages, web pages can be of any length.) Web pages sometimes are called *web documents*. Both terms refer to the same thing. A web page consists of an HTML document and all the other components that are included on the page, such as images or other media.

One problem with the term *home page* is that it means different things in different contexts. If you're browsing the Web, you usually can think of the home page as the web page that loads when you start your browser or when you click the Home button. Each browser has its own default home page, which generally leads to the website of the browser's creator or one that makes it some money through advertising when you visit.

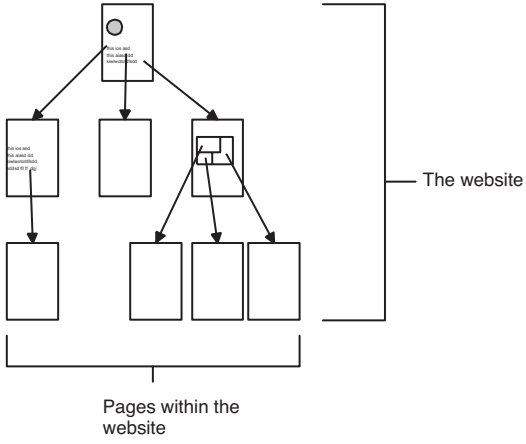


FIGURE 2.1
Websites and pages.

Within your browser, you can change that default home page to point to any page you want. Many users create a personalized page linking to sites they use often and set that as their browser's home page.

If you're publishing pages on the Web, however, the term *home page* has an entirely different meaning. The home page is the first or topmost page on your website. It's the intended entry point that provides access to the rest of the pages you've created (see Figure 2.2).

CAUTION

Most of your users will access your site through your home page, but some will enter your site through other pages. The nature of the Web is that people can link to any page on your site. If you have interesting information on a page other than your home page, people will link directly to that page. On the other pages of your site, you shouldn't assume that the visitor has seen your home page.

A home page usually contains an overview of the content of the website, available from that starting point—for example, in the form of a table of contents or a set of icons. If your content is small enough, you might include everything on that single home page—making your home page and your website the same thing. A personal home page might include a link to a person's resumé and some pictures from a recent vacation. A corporate home page usually describes what the company does and contains links like "About the Company," "Products and Services," and "Customer Support."

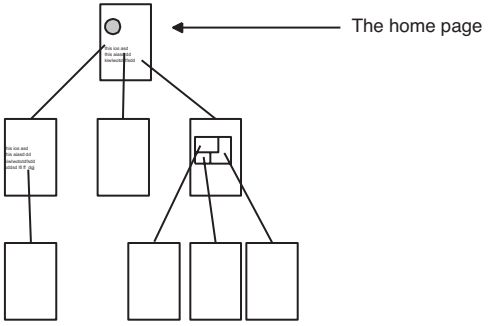


FIGURE 2.2
A home page.

What Do You Want to Do on the Web?

This question might seem silly. You wouldn't have bought this book if you didn't already have some idea of what you want to put online. But maybe you don't really know what you want to put on the Web, or you have a vague idea but nothing concrete. Maybe it has suddenly become your job to work on the company website, and someone handed you this book and said, "Here, this will help." Maybe you just want to do something similar to some other web page you've seen and thought was particularly cool.

What you want to put on the Web is what I refer to throughout this book as your content. *Content* is a general term that can refer to text, graphics, media, interactive forms, and so on. If you tell someone what your web pages are about, you're describing your content.

What sort of content can you put on the Web? Just about anything you want to. Here are some of the types of content that are popular on the Web right now:

- **Stuff for work**—Perhaps you work in the accounting department and you need to publish the procedure for filing expense reports on your company's intranet. Or you're a software developer and you need to publish the test plan for your company's next software release on an internal web server. Chances are that you can publish some information on a web page at work that will save you from having to type it into an email every time someone asks you about it. Try it!
- **Personal information**—You can create pages describing everything anyone could ever want to know about you and how incredibly marvelous you are—your hobbies, your resumé, your picture, things you've done.
- **Blogs and journals**—Many people publish their journals or their opinions on a blog. Many people use content management applications to publish their journals or blogs, but knowing HTML is still helpful for changing the look and feel of your site and sprucing up your individual entries or articles.

- **Hobbies or special interests**—A web page can contain information about a particular topic, hobby, or something you're interested in; for example, music, *Star Trek*, motorcycles, cult movies, hallucinogenic mushrooms, antique ink bottles, or upcoming jazz concerts in your city.
- **Publications**—Newspapers, magazines, and other publications lend themselves particularly well to the Web, and websites have the advantage of being more immediate and easier to update than their print counterparts. Delivery is a lot simpler, too. The same holds true for a newsletter for your garden club or news about your neighborhood association.
- **Company profiles**—You could offer information about what a company does, where it's located, job openings, data sheets, whitepapers, marketing collateral, product demonstrations, and whom to contact.
- **Online documentation**—The term *online documentation* can refer to everything from quick-reference cards to full reference documentation to interactive tutorials or training modules. Anything task-oriented (changing the oil in your car, making a soufflé, creating landscape portraits in oil, learning HTML) could be described as online documentation.
- **Shopping catalogs**—If your company offers items for sale, making your products available on the Web is a quick and easy way to let your customers know what you have available and your prices. If prices change, you can just update your web documents to reflect that new information.
- **Online stores**—The Web is a great place to sell things. Various sites let just about anybody sell stuff online. You can auction your goods off at eBay or sell them for a fixed price at half.com. Amazon.com lets you do both. You can also create your own online store if you want. There's plenty of software out there these days to make the task of selling things online a lot easier than it used to be.
- **Polling and opinion gathering**—Forms on the Web enable you to get feedback from your visitors via opinion polls, suggestion boxes, comments on your web pages or your products, or through interactive discussion groups.
- **Online education**—The low cost of information delivery to people anywhere with an Internet connection via the Web makes it an attractive medium for delivery of distance-learning programs. Already, numerous traditional universities, and new online schools and universities, have begun offering distance learning on the Web. For example, the Massachusetts Institute of Technology is placing teaching materials online for public use at http://ocw.mit.edu/.
- **Anything else that comes to mind**—Hypertext fiction, online toys, media archives, collaborative art…anything!

The only thing that limits what you can publish on the Web is your own imagination. If what you want to do with it isn't in this list or seems especially wild or half-baked, that's an excellent reason to try it. The most interesting web pages are the ones that stretch the boundaries of what the Web is supposed to be capable of.

You might also find inspiration in looking at other websites similar to the one you have in mind. If you're building a corporate site, look at the sites belonging to your competitors and see what they have to offer. If you're working on a personal site, visit sites that you admire and see whether you can find inspiration for building your own site. Decide what you like about those sites and you want to emulate, and where you can improve on those sites when you build your own.

If you really have no idea of what to put up on the Web, don't feel that you have to stop here; put this book away, and come up with something before continuing. Maybe by reading through this book, you'll get some ideas. (And this book will be useful even if you don't have ideas.) I've personally found that the best way to come up with ideas is to spend an afternoon browsing on the Web and exploring what other people have done.

Setting Your Goals

What do you want people to accomplish on your website? Are your visitors looking for specific information on how to do something? Are they going to read through each page in turn, going on only when they're done with the page they're reading? Are they just going to start at your home page and wander aimlessly around, exploring your world until they get bored and go somewhere else?

Suppose that you're creating a website that describes the company where you work. Some people visiting that website might want to know about job openings. Others might want to know where the company actually is located. Still others might have heard that your company makes technical whitepapers available over the Net, and they want to download the most recent version of a particular paper. Each of these goals is valid, so you should list each one.

For a shopping catalog website, you might have only a few goals: to enable your visitors to browse the items you have for sale by name or price, and to order specific items after they finish browsing.

For a personal or special-interest website, you might have only a single goal: to enable your visitors to browse and explore the information you provide.

The goals do not have to be lofty ("this website will bring about world peace") or even make much sense to anyone except you. Still, coming up with goals for your Web documents prepares you to design, organize, and write your web pages specifically to reach these goals. Goals also help you resist the urge to obscure your content with extra information.

If you're designing web pages for someone else—for example, if you're creating the website for your company or if you've been hired as a consultant—having a set of goals for the site from your employer definitely is one of the most important pieces of information you should have before you create a single page. The ideas you have for the website might not be the ideas that other people have for it, and you might end up doing a lot of work that has to be thrown away.

Breaking Up Your Content into Main Topics

With your goals in mind, try to organize your content into main topics or sections, chunking related information together under a single topic. Sometimes the goals you came up with in the preceding section and your list of topics will be closely related. For example, if you're putting together a web page for a bookstore, the goal of ordering books fits nicely under a topic called, appropriately, "Ordering Books."

You don't have to be exact at this point in development. Your goal here is just to try to come up with an idea of what, specifically, you'll be describing in your web pages. You can organize the information better later, as you write the actual pages.

Suppose that you're designing a website about how to tune up your car. This example is simple because tune-ups consist of a concrete set of steps that fit neatly into topic headings. In this example, your topics might include the following:

- Change the oil and oil filter.
- Check and adjust engine timing.
- Check and adjust valve clearances.
- Check and replace the spark plugs.
- Check fluid levels, belts, and hoses.

Don't worry about the order of the steps or how you're going to get your visitors to go from one section to another. Just list the points you want to describe in your website.

How about a less task-oriented example? Suppose that you want to create a set of web pages about a particular rock band because you're a big fan, and you're sure other fans would benefit from your extensive knowledge. Your topics might be as follows:

- The history of the band
- Biographies of each of the band members
- A *discography* (all the albums and singles the band has released)
- Selected lyrics
- Images of album covers
- Information about upcoming shows and future albums

You can come up with as many topics as you want, but try to keep each topic reasonably short. If a single topic seems too large, try to break it up into subtopics. If you have too many small topics, try to group them together into a more general topic heading. For example, if you're creating an online encyclopedia of poisonous plants, having individual topics for each plant would be overkill. You can just as easily group each plant name under a letter of the alphabet (A, B, C, and so on) and use each letter as a topic. That's assuming, of course, that your visitors will be looking up information in your encyclopedia alphabetically. If they want to look up poisonous plants by using some other method, you'd need to come up with another system of organization, too.

Your goal is to have a set of topics that are roughly the same size and that group together related bits of information you have to present.

Ideas for Organization and Navigation

At this point, you should have a good idea of what you want to talk about and a list of topics. The next step is to actually start structuring the information you have into a set of web pages. Before you do that, however, consider some standard structures that have been used in other help systems and online tools. This section describes some of these structures, their various features, and some important considerations, including the following:

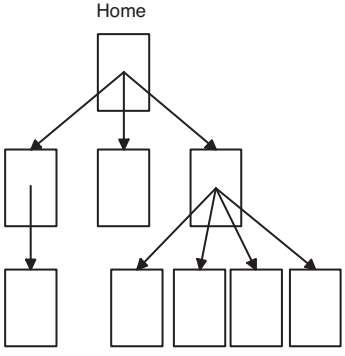
- The kinds of information that work well for each structure
- How visitors find their way through the content of each structure type to find what they need
- How to make sure that visitors can figure out where they are within your documents (context) and find their way back to a known position

As you read this section, think about how your information might fit into one of these structures or how you could combine these structures to create a new structure for your website.

Hierarchies

Probably the easiest and most logical way to structure your web documents is in a hierarchical or menu fashion, as illustrated in Figure 2.3. Hierarchies and menus lend themselves especially well to online and hypertext documents. Most online help systems, for example, are hierarchical. You start with a list or menu of major topics; selecting one leads you to a list of subtopics, which then leads you to a discussion about a particular topic. Different help systems have different levels, of course, but most follow this simple structure.

FIGURE 2.3
Hierarchical organization.



In a hierarchical organization, visitors can easily see their position in the structure. Their choices are to move up for more general information or down for more specific information. If you provide a link back to the top level, your visitors can get back to some known position quickly and easily.

In hierarchies, the home page provides the most general overview to the content below it. The home page also defines the main links for the pages farther down in the hierarchy. For example, a website about gardening might have a home page with the topics shown in Figure 2.4.

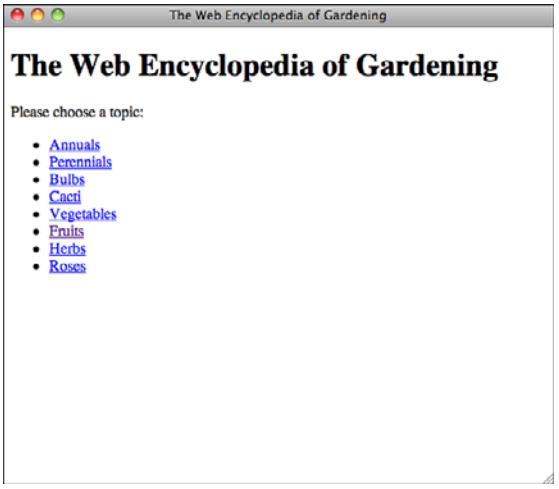


FIGURE 2.4
A Gardening home page with a hierarchical structure.

If you select Fruits, you then follow a link "down" to a page about fruits (see Figure 2.5). From there, you can go back to the home page, or you can select another link and go farther down into more specific information about particular fruits.

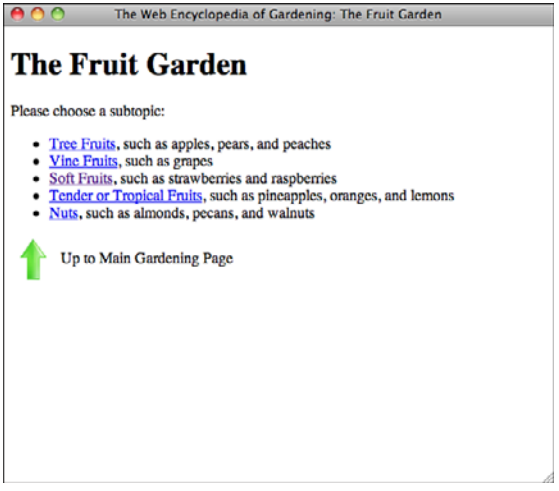


FIGURE 2.5
Your hierarchy takes you to the Fruits page.

Selecting Soft Fruits takes you to yet another menu-like page, where you have still more categories from which to choose (see Figure 2.6). From there, you can go up to Fruits, back to the home page, or down to one of the choices in this menu.

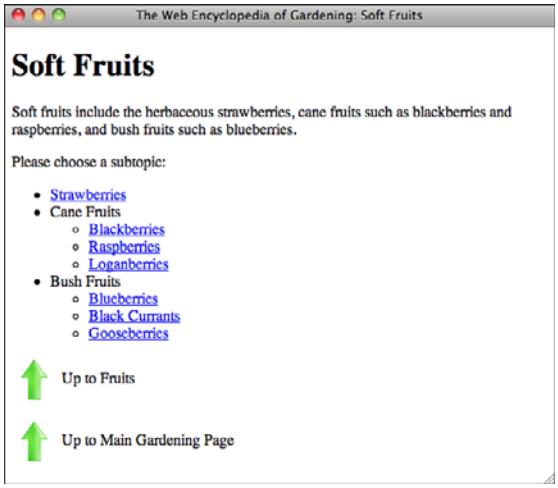


FIGURE 2.6
From the Fruits page, you can find the Soft Fruits page.

Note that each level has a consistent interface (up, down, back to index), and that each level has a limited set of choices for basic navigation. Hierarchies are structured enough that the chance of getting lost is minimal. This especially is true if you provide clues about where up is; for example, an Up to Soft Fruits link as opposed to just Up.

In addition, if you organize each level of the hierarchy and avoid overlap between topics (and the content you have lends itself to a hierarchical organization), using hierarchies can be an easy way to find particular bits of information. If that use is one of your goals for your visitors, using a hierarchy might work particularly well.

Avoid including too many levels and too many choices, however, because you can easily annoy your visitors. Having too many menu pages results in "voicemail syndrome." After having to choose from too many menus, visitors might forget what they originally wanted, and they're too annoyed to care. Try to keep your hierarchy two to three levels deep, combining information on the pages at the lowest levels (or endpoints) of the hierarchy if necessary.

Linear

Another way to organize your documents is to use a linear or sequential organization, similar to how printed documents are organized. In a linear structure, as illustrated in Figure 2.7, the home page is the title or introduction, and each page follows sequentially. In a strict linear structure, links move from one page to another, typically forward and back. You also might want to include a link to Home that takes the user quickly back to the first page.

www.allitebooks.com

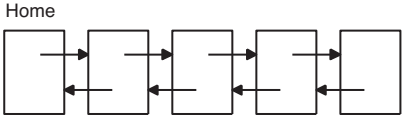


FIGURE 2.7 Linear organization.

Context generally is easy to figure out in a linear structure simply because there are so few places to go.

A linear organization is rigid and limits your visitors' freedom to explore and your freedom to present information. Linear structures are good for putting material online when the information also has a linear structure offline (such as short stories, step-by-step instructions, or computer-based training), or when you explicitly want to prevent your visitors from skipping around.

For example, consider teaching someone how to make cheese by using the Web. Cheese making is a complex process involving several steps that must be followed in a specific order.

Describing this process using web pages lends itself to a linear structure rather well. When navigating a set of web pages on this subject, you'd start with the home page, which might have a summary or an overview of the steps to follow. Then, by using the link for going forward, move on to the first step, Choosing the Right Milk; to the next step, Setting and Curdling the Milk; all the way through to the last step, Curing and Ripening the Cheese. If you need to review at any time, you could use the link for moving backward. Because the process is so linear, you would have little need for links that branch off from the main stem or links that join together different steps in the process.

NOTE

The linear navigation style is most commonly seen with long articles on newspaper and magazine websites. The articles are often split into multiple pages, and navigation is provided to make it easy to move through the pages of the article sequentially.

Linear with Alternatives

You can soften the rigidity of a linear structure by enabling the visitors to deviate from the main path. You could, for example, have a linear structure with alternatives that branch out from a single point (see Figure 2.8). The offshoots can then rejoin the main branch at some point farther down, or they can continue down their separate tracks until they each come to an end.

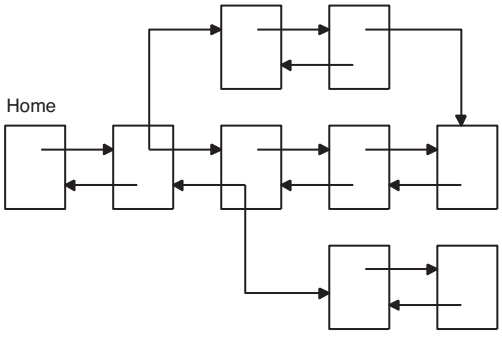


FIGURE 2.8
Linear with alternatives.

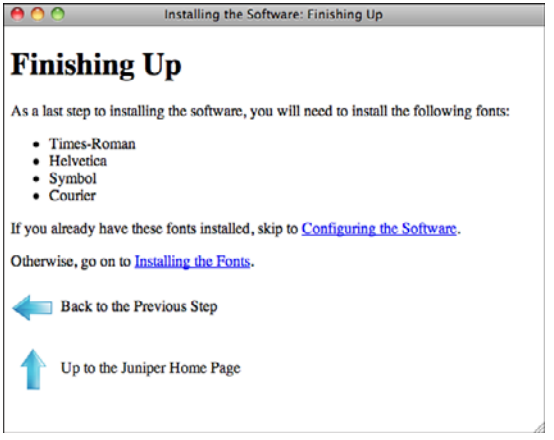
Suppose that you have an installation procedure for a software package that's similar in most ways, regardless of the computer type, except for one step. At that point in the linear installation, you could branch out to cover each system, as shown in Figure 2.9.

FIGURE 2.9
Different steps for different systems.

After the system-specific part of the installation, you could link back to the original branch and continue with the generic installation.

In addition to branching from a linear structure, you could also provide links that enable visitors to skip forward or backward in the chain if they need to review a particular step or if they already understand some content (see Figure 2.10).

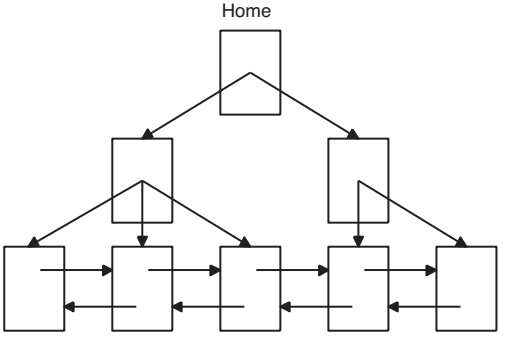
FIGURE 2.10
Skip ahead or back.



Combination of Linear and Hierarchical

A popular form of document organization on the Web is a combination of a linear structure and a hierarchical one, as shown in Figure 2.11. This structure occurs most often when documents that are both linear and structured are published online. This approach is often seen with reference manuals on the Web. Pages include links to the next and previous sections for readers who are moving linearly through the manual, and links up through the document's hierarchy, represented by the table of contents, are included, too.

FIGURE 2.11
Combination of linear and hierarchical organization.

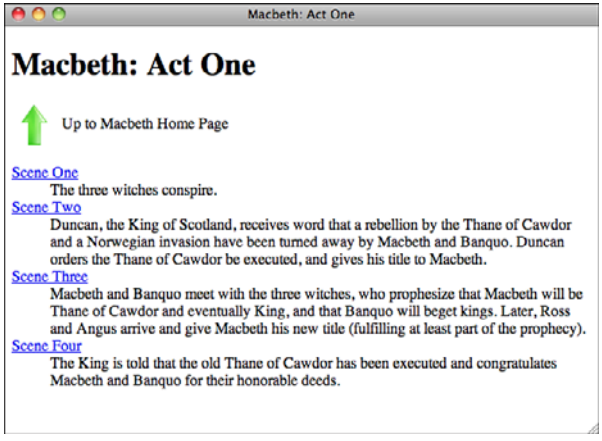


The combination of linear and hierarchical documents works well as long as you have appropriate clues regarding context. Because the visitors can either move up and down or forward and backward, they can easily lose their mental positioning in the hierarchy when crossing hierarchical boundaries by moving forward or backward.

Suppose that you're putting the Shakespeare play *Macbeth* online as a set of web pages. In addition to the simple linear structure that the play provides, you can create a

hierarchical table of contents and summary of each act linked to appropriate places within the text, similar to what is shown in Figure 2.12.

FIGURE 2.12
Macbeth's hierarchy.



Because this structure is both linear and hierarchical, you provide links to go forward and backward, and links to return to the beginning and to move up in the hierarchy. But what is the context for going up?

If you've just come down into this page from an act summary, the context makes sense: *Up* means go back to the summary from which you just came.

But suppose that you go down from a summary and then go forward, crossing an act boundary (say from Act 1 to Act 2). Now what does *up* mean? The fact that you're moving up to a page you might not have seen before could be disorienting given the nature of what you expect from a hierarchy. Up and down are supposed to be consistent.

Consider two possible solutions:

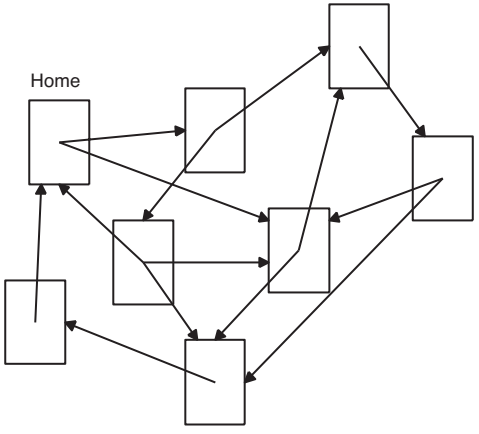
- Do not allow forward and back links across hierarchical boundaries. In this case, to read from Act 1 to Act 2 in *Macbeth*, you have to move up in the hierarchy and then back down into Act 2.
- Provide more context in the link text. Rather than just Up or an icon for the link that moves up in the hierarchy, include a description of where the user is moving to.

Web

A web is a set of documents with little or no actual overall structure; the only thing tying each page together is a link (see Figure 2.13). Visitors drift from document to document, following the links around.

For an example of such a site, visit Wikipedia at http://wikipedia.org. Wikipedia is an encyclopedia written and maintained by the public. Anyone can write a new article or edit an existing article, and the site is loosely organized. Articles that reference topics discussed in other articles link to them, creating a web organization scheme. Wikipedia has no hierarchical organization; you're expected to find the topics you're interested in by following links or using the site's search functionality.

FIGURE 2.13
A web structure.



Web structures tend to be free-floating and enable visitors to wander aimlessly through the content. Web structures are excellent for content that's intended to be meandering or unrelated or when you want to encourage browsing. The World Wide Web itself is, of course, a giant web structure.

In the context of a website, the environment is organized so that each page is a specific location (and usually contains a description of that location). From that location, you can move in several different directions, exploring the environment much in the way you would move from room to room in a building in the real world (and getting lost just as easily). The initial home page, for example, might look something like the one shown in Figure 2.14.

From that page, you then can explore one of the links, for example, to go into the building, which takes you to the page shown in Figure 2.15.

FIGURE 2.14
The Wikipedia home page. Some hierarchy is imposed.

FIGURE 2.15
A Wikipedia article page, with links to related topics interspersed throughout.

Each room has a set of links to each adjacent room in the environment. By following the links, you can explore the rooms in the environment.

The problem with web organizations is that you can get easily lost in them—just as you might in the world you're exploring in the example. Without any overall structure to the content, figuring out the relationship between where you are, where you're going and, often, where you've been, is difficult. Context is difficult, and often the only way to find your way back out of a web structure is to retrace your steps. Web structures can be extremely disorienting and immensely frustrating if you have a specific goal in mind.

To solve the problem of disorientation, you can use clues on each page. Here are two ideas:

- Provide a way out. Return to Home Page is an excellent link.
- Include a map of the overall structure on each page, with a "you are here" indication somewhere in the map. It doesn't have to be an actual visual map, but providing some sort of context goes a long way toward preventing your visitors from getting lost.

Storyboarding Your Website

The next step in planning your website is to figure out what content goes on what page and to come up with some simple links for navigation between those pages.

If you use one of the structures described in the preceding section, much of the organization might arise from that structure—in which case, this section will be easy. However, if you want to combine different kinds of structures or if you have a lot of content that needs to be linked together in sophisticated ways, sitting down and making a specific plan of what goes where can be incredibly useful later as you develop and link each individual page.

What's Storyboarding and Why Do I Need It?

Storyboarding a website is a concept borrowed from filmmaking in which each scene and each individual camera shot is sketched and roughed out in the order in which it occurs in the movie. Storyboarding provides an overall structure and plan to the film that enables the director and staff to have a distinct idea of where each individual shot fits into the overall movie.

The storyboarding concept works quite well for developing web pages. The storyboard provides an overall rough outline of what the website will look like when it's done, including which topics go on which pages, the primary links, and maybe even some conceptual idea of what sort of graphics you'll be using and where they'll go. With that representation in hand, you can develop each page without trying to remember exactly where that page fits into the overall website and its often complex relationships to other pages.

In the case of large sets of documents, a storyboard enables different people to develop various portions of the same website. With a clear storyboard, you can minimize duplication of work and reduce the amount of contextual information each person needs to remember.

For smaller or simpler websites, or websites with a simple logical structure, storyboarding might be unnecessary. For larger and more complex projects, however, the existence of a storyboard can save enormous amounts of time and frustration. If you can't keep all the parts of your content and their relationships in your head, consider creating a storyboard.

So, what does a storyboard for a website look like? It can be as simple as a couple of sheets of paper. Each sheet can represent a page, with a list of topics each page will describe and some thoughts about the links that page will include. I've seen storyboards for complex hypertext systems that involved a large bulletin board, index cards, and string. Each index card had a topic written on it, and the links were represented by string tied on pins from card to card.

The point of a storyboard is that it organizes your web pages in a way that works for you. If you like index cards and string, work with these tools. If a simple outline on paper or on the computer works better, use that instead.

Hints for Storyboarding

Some things to think about when developing your storyboard are as follows:

- **Which topics will go on each page?**

  A simple rule of thumb is to have each topic represented by a single page. If you have several topics, however, maintaining and linking them can be a daunting task. Consider combining smaller, related topics onto a single page instead. Don't go overboard and put everything on one page, however; your visitors still have to download your document over the Internet. Having several medium-sized pages (such as the size of 2 to 10 pages in your word processor) is better than having one monolithic page or hundreds of little tiny pages.

- **What are the primary forms of navigation between pages?**

  What links will you need for your visitors to navigate from page to page? They are the main links in your document that enable your visitors to accomplish the goals you defined in the first section. Links for forward, back, up, down, and home all fall under the category of primary navigation.

- **What alternative forms of navigation are you going to provide?**

  In addition to the simple navigation links, some websites contain extra information that's parallel to the main web content, such as a glossary of terms, an alphabetic index of concepts, copyright information, or a credits page. Consider these extra forms of information when designing your plan, and think about how you're going to link them into the main content.

- **What will you put on your home page?**

  Because the home page is the starting point for the rest of the information in your website, consider what sort of information you're going to put on the home page. A blog? A general summary of what's to come? A list of links to other topics? Whatever you put on the home page, make sure that it's compelling enough so that members of your intended audience want to stick around.

- **What are your goals?**

  As you design the framework for your website, keep your goals in mind, and make sure that you aren't obscuring your goals with extra information or content.

TIP

Several utilities and packages can assist you in storyboarding. Foremost among them are site-management packages that can help you manage links in a site, view a graphical representation of the relationship of documents in your site, move documents around, and automatically update all relevant links in and to the documents. This type of functionality is often provided by content-management applications for websites.

Web Hosting

At some point, you'll want to move the websites you create from your local computer to a server on the Internet. Before doing so, you have to decide exactly what kind of hosting arrangement you want. The simplest approach is to get a Web hosting account that enables you to upload your HTML files, images, style sheets, and other web content to a server that's visible on the Web. This approach enables you to easily create web pages (and websites) locally and publish them on the server without making changes to them.

Using a Content-Management Application

The other option is to use an application to publish content on the Web. This can make more sense if your idea for a website falls into an existing category with publishing tools available for it. For example, if you want to publish a blog, you can use sites like TypePad (http://typepad.com/), Blogger (http://blogger.com), WordPress (http://wordpress.com/), Tumblr (http://tumblr.com) or Posterous (http://posterous.com), among many others. The advantage of these applications is that it's easy to set up a site, pick a theme, and start publishing content on the Web through a web interface. There's no need to build the web pages by hand, set up a hosting account, or even deal with editing files by hand.

Similarly, applications are available for creating other kinds of sites, too. If you want to create a social site for a group, you can use applications like Ning (http://ning.com) or Drupal Gardens (http://www.drupalgardens.com/). You can create a free wiki-type site (like Wikipedia) at wikidot (http://www.wikidot.com/) or PBworks (http://pbworks.com/).

Generally with these types of applications, all you need to do to start is fill out a form, choose a URL, and pick a theme for your website. Then you can enter your content by way of forms, enabling you to avoid writing the HTML for the pages. Some of them even include WYSIWYG editors so that you can format the content you enter without using HTML.

However, that doesn't mean that you don't need to learn anything about HTML or *Cascading Style Sheets (CSS)*. Even if you're not creating the pages by hand, you'll still need to understand how pages are structured when you start entering content or modifying themes. If you don't understand how Web pages are built, you won't know how to track down and fix problems with the markup on your website, whether you're responsible for writing it.

For most people taking their first steps into web publishing, using an application to get started is often the best approach, because it enables you to start putting the content you're interested in on the Web immediately without figuring out too many things for yourself. However, often people run into limitations in these applications that leave them wanting to take more control of their websites and go further on their own. This book can help you do so.

Setting Up Your Own Web Hosting

If you do want to create and upload your own web pages, you need to choose a company that can provide you with the space you need. There are a huge number of hosting companies who provide web space to people who want to launch their own websites. Companies like DreamHost (http://dreamhost.com/) and Pair.com (http://pair.com) have been in the hosting business for many years and offer a variety of affordable hosting plans, but there are plenty of other options, too. Many people subscribe to hosting plans from the company that they use to register the domain names for their websites, or go with hosting companies that are in their local area.

If you choose to go this route, the steps for going from setting up a hosting account to making your pages available on the Web are as follows:

www.allitebooks.com

1. Optionally, register a domain name. If you want your website to appear at a URL like mycoolsite.com or mycompany.com, you need to register that domain name if you haven't already. There are a number of domain registrars, just enter domain registration in your favorite search engine to see a large number of ads and search results for companies that offer domain registration.
2. Pick out a web hosting company and sign up for an account. It may be the case that you don't need to do this. If you're going to be putting your pages on an internal or external server belonging to your employer or your school, you won't need your own hosting. But if you're creating a new website that will be available on the Internet, you need some sort of hosting arrangement.
3. Associate your domain name with your new website, if you have registered one. Your domain registrar and hosting company should provide instructions for setting it up so that your domain name points to your hosting account. That way when users enter your domain name in a URL, they'll get the content that you upload to your server.
4. Start uploading your content. When your web hosting is set up, you can use whatever tool you prefer to start uploading web content to the server. Many hosts provide a web interface that can allow you to upload content, but usually it's much easier to use a file transfer tool that supports File Transfer Protocol (FTP), Secure Copy (SCP), or Secure FTP (SFTP) to get your files to the server.

There will be a more extensive discussion of web hosting and how to publish your site in Lesson 20, "Putting Your Site Online," but I wanted to give you a head start if you're eager to start publishing on the Web.

Summary

Designing a website, like designing a book outline, a building plan, or a painting, can sometimes be a complex and involved process. Having a plan before you begin can help you keep the details straight and help you develop the finished product with fewer false starts. In this lesson, you learned how to put together a simple plan and structure for creating a set of web pages, including the following:

- Deciding what sort of content to present
- Coming up with a set of goals for that content
- Deciding on a set of topics
- Organizing and storyboarding the website
- Publishing your site on the Web

With that plan in place, you now can move on to the next few lessons and learn the specifics of how to write individual web pages, create links between them, and add graphics and media to enhance the website for your audience.

Workshop

The first section of the workshop lists some of the common questions people ask while planning a website, along with an answer to each. Following that, you have an opportunity to answer some quiz questions. If you have problems answering any of the questions in the quiz, go to the next section, where you can find the answers. The exercises help you formulate some ideas for your own website.

Q&A

Q **Getting organized seems like an awful lot of work. All I want to do is make something simple, and you're telling me I have to have goals and topics and storyboards. Are all the steps listed here really necessary?**

A If you're doing something simple, you won't need to do much, if any, of the stuff I recommended in this lesson. However, if you're talking about developing two or three interlinked pages or more, having a plan before you start can help. If you just dive in, you might discover that keeping everything straight in your head is too difficult. And the result might not be what you expected, making it hard for people to get the information they need out of your website and making it difficult for you to reorganize it so that it makes sense. Having a plan before you start can't hurt, and it might save you time in the long run.

Q **You talked a lot in this lesson about organizing topics and pages, but you said nothing about the design and layout of individual pages. Why?**

A I discuss design and layout later in this book, after you've learned more about the sorts of layout that HTML (the language used for web pages) can do and the stuff that it just can't do. You'll find a whole day and more about page layout and design in Lesson 18, "Writing Good Web Pages: Do's and Don'ts."

Q **What if I don't like any of the basic structures you talked about in this lesson?**

A Then design your own. As long as your visitors can find what they want or do what you want them to do, no rules say you *must* use a hierarchy or a linear structure. I presented these structures only as potential ideas for organizing your web pages.

Quiz

1. How would you briefly define the meaning of the terms *website*, *web server*, and *web pages*?
2. In terms of web publishing, what's the meaning of the term *home page*?
3. After you set a goal or purpose for your website, what's the next step to designing your pages?
4. Regardless of the navigation structure you use in your website, there's one link that should typically appear on each of your web pages. What is it?
5. What's the purpose of a storyboard?

Quiz Answers

1. A *website* is one or more web pages linked together in a meaningful way. A *web server* is the actual computer that stores the website (or confusingly enough, the piece of software that responds to requests for pages from the browser). *Web pages* are the individual elements of the website, like a page is to a book.
2. A *home page*, in terms of web publishing, is the entry point to the rest of the pages in your website (the first or topmost page).
3. After you set a goal or purpose for your website, you should try to organize your content into topics or sections.
4. You should try to include a link to your home page on each of the pages in your website. That way, users can always find their way back home if they get lost.
5. A storyboard provides an overall outline of what the website will look like when it's done. It helps organize your web pages in a way that works for you. They are most beneficial for larger websites.

Exercises

1. Come up with a list of several goals that your visitors might have for your web pages. The clearer your goals, the better.
2. After you set your goals, visit sites on the Web that cover topics similar to those you want to cover in your own website. As you examine the sites, ask yourself whether they're easy to navigate and have good content. Then make a list of what you like about the sites. How would you make your website better?

LESSON 3
Introducing HTML and XHTML

After finishing up the discussions about the World Wide Web and getting organized, with a large amount of text to read and concepts to digest, you're probably wondering when you're actually going to get to write a web page. That is, after all, why you bought this book. Wait no longer! In this lesson, you get to create your first (albeit brief) web page, learn about HTML (the language for writing web pages), and learn about the following:

- What HTML is and why you have to use it
- What you can and cannot do when you design HTML pages
- What HTML tags are and how to use them
- How to write pages that conform to the XHTML standard
- How you can use Cascading Style Sheets to control the look and feel of your pages

What HTML Is (And What It Isn't)

Take note of just one more thing before you start actually writing web pages. You should know what HTML is, what it can do, and most important, what it can't do.

HTML stands for *Hypertext Markup Language*. HTML was based on the *Standard Generalized Markup Language (SGML)*, a much larger, more complicated document-processing system. To write HTML pages, you won't need to know much about SGML. However, knowing that one of the main features of SGML is that it describes the general structure of the content inside documents—rather than its actual appearance on the page or onscreen—does help. This concept might be a bit foreign to you if you're used to working with WYSIWYG (What You See Is What You Get) editors such as Adobe's Dreamweaver or Microsoft FrontPage, so let's go over the information carefully.

HTML Describes the Structure of a Page

HTML, by virtue of its SGML heritage, is a language for describing the structure of a document, not its actual presentation. The idea here is that most documents have common elements—for example, titles, paragraphs, and lists. Before you start writing, therefore, you can identify and define the set of elements in that document and name them appropriately (see Figure 3.1).

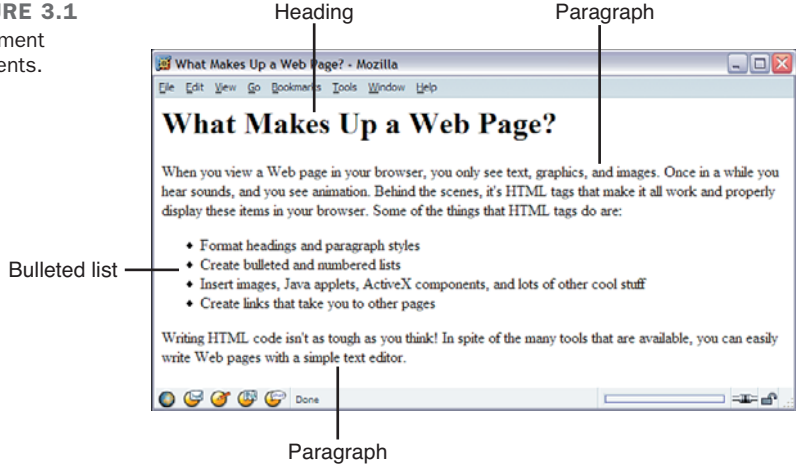


FIGURE 3.1 Document elements.

If you've worked with word processing programs that use style sheets (such as Microsoft Word) or paragraph catalogs (such as FrameMaker), you've done something similar; each section of text conforms to one of a set of styles that are predefined before you start working.

HTML defines a set of common styles for web pages: headings, paragraphs, lists, and tables. It also defines character styles such as boldface and code examples. These styles are indicated inside HTML documents using *tags*. Each tag has a specific name and is set off from the content of the document using a notation that I discuss a bit later.

HTML Does Not Describe Page Layout

When you work with a word processor or page layout program, styles are not just named elements of a page; they also include formatting information such as the font size and style, indentation, underlining, and so on. So, when you write some text that's supposed to be a heading, you can apply the Heading style to it, and the program automatically formats that paragraph for you in the correct style.

HTML doesn't go this far. For the most part, the HTML specification doesn't say anything about how a page looks when it's viewed. HTML tags just indicate that an element is a heading or a list; they say nothing about how that heading or list is to be formatted. So, as with the magazine example and the layout person who formats your article, the layout person's job is to decide how big the heading should be and what font it should be in. The only thing you have to worry about is marking which section is supposed to be a heading.

NOTE

Although HTML doesn't say much about how a page looks when it's viewed, *Cascading Style Sheets (CSS)* enable you to apply advanced formatting to HTML tags. HTML has evolved to the point where web publishers are intended to use CSS for formatting instructions. I'll talk about CSS later in this lesson.

Web browsers, in addition to providing the networking functions to retrieve pages from the Web, double as HTML formatters. When you read an HTML page into a browser such as Firefox, Safari, or Internet Explorer, the browser interprets, or *parses*, the HTML tags and formats the text and images on the screen. The browser has mappings between the names of page elements and actual styles on the screen; for example, headings might be in a larger font than the text on the rest of the page. The browser also wraps all the text so that it fits into the current width of the window.

Different browsers running on diverse platforms might style elements differently. Some browsers might use different font styles than others. For example, a browser on a desktop computer might display italics as italics, whereas a handheld device or mobile phone might use reverse text or underlining on systems that don't have italic fonts. Or it might put a heading in all capital letters instead of a larger font.

What this means to you as a web page designer is that the pages you create with HTML might look different from system to system and from browser to browser. The actual information and links inside those pages are still there, but the onscreen appearance changes. You can design a web page so that it looks perfect on your computer system, but when someone else reads it on a different system, it might look entirely different. (And it might vbe entirely unreadable.)

How the Visual Styles for Tags Evolved

In practice, most HTML tags are rendered in a fairly standard manner, on desktop computers at least. When the earliest browsers were written, somebody decided that links would be underlined and blue, visited links would be purple, and emphasized text would appear in italic. They also made similar decisions about every other tag. Since then, pretty much every browser maker has followed that convention to a greater or lesser degree. These conventions blurred the line separating structure from presentation, but in truth it still exists, even if it's not obvious.

Why It Works This Way

If you're used to writing and designing documents that will wind up printed on paper, this concept might seem almost perverse. No control over the layout of a page? The whole design can vary depending on where the page is viewed? This is awful! Why on earth would a system work like this?

Remember in Lesson 1, "Navigating the World Wide Web," when I mentioned that one of the cool things about the Web is that it's cross-platform and that web pages can be viewed on any computer system, on any size screen, with any graphics display? If the final goal of web publishing is for your pages to be readable by anyone in the world, you can't count on your readers having the same computer systems, the same screen size, the same number of colors, or the same fonts that you have. The Web takes into account all these differences and enables all browsers and all computer systems to be on equal ground.

The Web, as a design medium, is not a new form of paper. The Web is an entirely different medium, with its own constraints and goals that are different from working with paper. The most important rules of web page design, as I'll keep harping on throughout this book, are the following:

DO	DON'T
Do design your pages so that they work in most browsers.	**Don't** design your pages based on what they look like on your computer system and on your browser.
Do focus on clear, well-structured content that's easy to read and understand.	

Throughout this book, I'll show you examples of HTML code and what they look like when displayed.

How Markup Works

HTML is a *markup language*. Writing in a markup language means that you start with the text of your page and add special tags around words and paragraphs. The tags indicate the different parts of the page and produce different effects in the browser. You learn more about tags and how they're used in the next section.

HTML has a defined set of tags you can use. You can't make up your own tags to create new styles or features. And just to make sure that things are confusing, various browsers support different sets of tags. To further understand this, take a brief look at the history of HTML.

A Brief History of HTML Tags

HTML 2.0 was the original standard for HTML (a written specification for it is developed and maintained by the W3C) and the set of tags that all browsers must support. Most of the tags in that original specification are still supported. In the next few lessons, you primarily learn to use tags that were first introduced in HTML 2.0.

The HTML 3.2 specification was developed in early 1996. Several software vendors, including IBM, Microsoft, Netscape Communications Corporation, Novell, SoftQuad, Spyglass, and Sun Microsystems, joined with the W3C to develop this specification. Some of the primary additions to HTML 3.2 included features such as tables, applets, and text flow around images.

NOTE

The enhancements introduced in HTML 3.2 are covered later in this book. You learn more about tables in Lesson 10, "Building Tables." Lesson 12, "Integrating Multimedia: Sound, Video, and More," tells you how to use Java applets.

HTML 4.0, first introduced in 1997, incorporated many new features that gave designers greater control over page layout than HTML 2.0 and 3.2. Like HTML 2.0 and 3.2, the W3C maintains the HTML 4.0 standard.

Framesets (originally introduced in Netscape 2.0) and floating frames (originally introduced in Internet Explorer 3.0) became an official part of the HTML 4.0 specification. Framesets are discussed in more detail in Lesson 17, "Working with Frames and Linked Windows." We also see additional improvements to table formatting and rendering. By far, however, the most important change in HTML 4.0 was its increased integration with style sheets.

NOTE

If you're interested in how HTML development is working and just exactly what's going on at the W3C, check out the pages for HTML at the Consortium's site at http://www.w3.org/pub/WWW/MarkUp/.

At one time, Microsoft and Netscape were releasing new versions of their browsers frequently, competing to see who could add the most compelling new features to HTML without waiting for the standards process to catch up. These days, browser releases in the browser market are growing. Microsoft Internet Explorer and Mozilla Firefox are popular, and other browsers such as Apple Safari, Google Chrome, and Opera are in the mix, too. Although they release new versions frequently, all of them are focused on implementing web standards instead of introducing nonstandard features of their own. The most important recent development, however, has been the expansion of the Web onto mobile devices. Mobile phones and other devices are growing more powerful and popular, and it is becoming more important for web developers to consider these platforms when designing their browsers.

The extra work involved in dealing with variations between browsers and platforms has been a headache for web developers for a long time. Keeping track of all this information can be confusing. Throughout this book, as I introduce each tag, I explain any browser-specific issues you may encounter.

The Current Standard: XHTML 1.1

XHTML 1.1 is written in *Extensible Markup Language (XML)*, and is the current standard that most web developers adhere to. The *X* stands for XML, which is another markup standard derived from SGML. The main difference from HTML is that XML has strict rules for document structure. Whereas HTML 4 was forgiving of unclosed

elements, for an XML document to be valid, every tag must be closed, every attribute must have a value, and more. XHTML 1.1 requires that HTML documents also be valid XML.

Technically, XHTML 1.1 and HTML 4.01 are *very* similar. The tags and attributes are almost the same, but a few simple rules have to be followed to make sure that a document complies with the XHTML 1.1 specification. Throughout this book, I explain how to deal with the different HTML tags to ensure that your pages are readable and still look good in all kinds of browsers.

The Future Standard: HTML5

The W3C HTML Working Group is busy creating a new standard for HTML: HTML5. The goal of HTML5 is to introduce new elements that more accurately reflect the state of the Web as it exists now. These elements include things like `<header>` and `<footer>` for the page header and footer, respectively.

HTML5 does not demand that web pages be valid XML, relaxing some of the rules that XHTML 1.0 imposed. However, today's valid HTML or XHTML will still be valid in HTML5 when it's fully adopted.

Most browsers have already begun to support the new features in HTML5, even though the specification has not been finalized. I'll explain what you can do with HTML5 now and what you'll need to do as browser support for HTML5 expands.

One of the most important differences between HTML5 and earlier HTML specifications is that HTML5 is being created with the cooperation of the browser makers. In the past, support for HTML specifications among browser vendors has been uneven at best. Only features that the browser vendors have committed to support will be included in HTML5. If agreement cannot be reached on a particular feature, that feature will not be included in a specification. The hope is that at the end of the HTML5 process, web developers will have a specification they can count on to work from.

Finally, HTML5 removes many elements that had been introduced in previous standards but are now superseded by Cascading Style Sheets. Some of these tags were deprecated in previous standards—HTML5 drops them entirely.

What HTML Files Look Like

Pages written in HTML are plain text files (ASCII), which means that they contain no platform- or program-specific information. Any editor that supports text (which should

be just about any editor—more about this subject in "Programs to Help You Write HTML" section, later) can read them. HTML files contain the following:

- The text of the page itself
- HTML tags that indicate page elements, structure, formatting, and hypertext links to other pages or to included media

Most HTML tags look something like the following:

```
<thetagname>affected text</thetagname>
```

The tag name itself (here, **thetagname**) is enclosed in brackets (< >). HTML tags generally have a beginning and an ending tag surrounding the text they affect. The beginning tag "turns on" a feature (such as headings, bold, and so on), and the ending tag turns it off. Closing tags have the tag name preceded by a slash (/). The opening tag (for example, for paragraphs) and closing tag (for example, for paragraphs) compose what is officially called an *HTML element*.

CAUTION

Be aware of the difference between the forward slash (/) mentioned with relation to tags, and backslashes (\), which are used by DOS and Windows in directory references on hard drives (as in C:\window or other directory paths). If you accidentally use the backslash in place of a forward slash in HTML, the browser won't recognize the ending tags.

Not all HTML tags have both an opening and closing tag. Some tags are only one-sided, and still other tags are containers that hold extra information and text inside the brackets. XHTML 1.1, however, requires that *all* tags be closed. You'll learn the proper way to open and close the tags as the book progresses.

Another difference between HTML 4.0 and XHTML 1.1 relates to usage of lowercase tags and attributes. HTML tags are not case-sensitive; that is, you can specify them in uppercase, lowercase, or in any mixture. So, <HTML> is the same as <html>, which is the same as <HtMl>. This isn't the case for XHTML 1.1, where all tag and attribute names must be written in lowercase. To get you thinking in this mindset, the examples in this book display tag and attribute names in bold lowercase text.

Task: Exercise 3.1: Creating Your First HTML Page

Now that you've seen what HTML looks like, it's your turn to create your own web page. Start with a simple example so that you can get a basic feel for HTML.

To start writing HTML, you don't need a web server, a web host, or even a connection to the Web itself. All you need is an application in which you can create your HTML files and at least one browser to view them. You can write, link, and test whole suites of web pages without even touching a network. In fact, that's what you're going to do for the majority of this book. Later, I discuss publishing everything on the Web so that other people can see your work.

To start, you need a text editor. A *text editor* is a program that saves files in ASCII format. ASCII format is just plain text, with no font formatting or special characters. For Windows, Notepad and Microsoft WordPad are good basic text editors (and are installed by default). Shareware text editors are also available for various operating systems, including, Windows, Mac OS X, and Linux. If you point your web browser to http://www.download.com and enter Text Editors as a search term, you'll find many applications available to download. If you're a Windows user, you might want to check out HTML-Kit in particular. It's a free text editor specifically built for editing HTML files. You can download it from http://www.chami.com/html-kit/. By the same token, Mac users might want to look at TextWrangler, available from http://www.barebones.com. If you prefer to work in a word processor such as Microsoft Word, don't panic. You can still write pages in word processors just as you would in text editors, although doing so is more complicated. When you use the Save or Save As command, you'll see a menu of formats you can use to save the file. One of them should be Text Only, Text Only with Line Breaks, or DOS Text. All these options will save your file as plain ASCII text, just as if you were using a text editor. For HTML files, if you have a choice between DOS Text and just Text, use DOS Text, and use the Line Breaks option if you have it.

CAUTION

Many word processors are including HTML modes or mechanisms for creating HTML or XML code. This feature can produce unusual results or files that simply don't behave as you expect. Using a word processor to generate HTML is not a good idea if you plan on editing the web pages later. When you work on the examples in this book, you should use a regular text editor.

What about the plethora of free and commercial HTML editors that offer to help you write HTML more easily? Some are text editors that simplify common tasks associated with HTML coding. If you have one of these editors, go ahead and use it. If you have a fancier editor that claims to hide all the HTML for you, put it aside for the next couple of lessons and try using a plain text editor just for a little while. Appendix A, "Sources of Further Information," lists many URLs where you can download free and commercial HTML editors available for different platforms. They appear in the section titled "HTML Editors and Converters."

Open your text editor and type the following code. You don't have to understand what any of it means at this point. You learn more about much of this in this lesson and the following lesson. This simple example is just to get you started:

```
<!DOCTYPE html><html>
<head>
<title>My Sample HTML Page</title>
</head>
<body>
<h1>This is an HTML Page</h1>
</body>
</html>
```

NOTE

Note that the `<!DOCTYPE>` tag in the previous example doesn't appear in lowercase like the rest of the tags. This tag is an exception to the XHTML rule and should appear in uppercase. The purpose of the `DOCTYPE` is to tell validators and browsers which specification your page was written to, in this case, HTML5. In Lesson 19, "Designing for the Real World," I talk about other DOCTYPEs you'll see, and how to choose which one to use for your pages.

After you create your HTML file, save it to your hard disk. Remember that if you're using a word processor like Microsoft Word, choose Save As and make sure that you're saving it as Text Only. When you choose a name for the file, follow these two rules:

- The filename should have an extension of `.html` (`.htm` is OK, but not preferred)—for example, `myfile.html`, `text.html`, or `index.htm`. Most web software requires your files to have these extensions, so get into the habit of doing it now. (If you use Windows, make sure that your computer is configured to show file extensions. If it isn't, you'll find yourself creating files named things like `myfile.html.txt`, which your browser will not think are HTML files.)

- Use small, simple names. Don't include spaces or special characters (bullets, accented characters)—just letters and numbers are fine. Be sure to choose descriptive, readable names for your files. They'll help you keep track of what they're used for, and they can help make your site friendlier to search engines.

Task: Exercise 3.2: Viewing the Result

Now that you have an HTML file, start your web browser. You don't have to be connected to the Internet because you're not going to be opening pages at any other site. Your browser or network connection software might complain about the lack of a network connection, but you can work offline.

After your browser is running, look for a menu item or button labeled Open, Open File, or maybe Open Page. Choosing it enables you to browse your local disk. The Open command (or its equivalent) opens a document from your local disk, parses it, and displays it. By using your browser and the Open command, you can write and test your HTML files on your computer in the privacy of your own home. (On most operating systems, you can just drag the icon from your HTML file into an open browser window if you prefer.)

If you don't see something similar to what's shown in Figure 3.2 (for example, if parts are missing or if everything looks like a heading), go back into your text editor and compare your file to the example. Make sure that all your tags have closing tags and that all your < characters are matched by > characters. You don't have to quit your browser to do so; just fix the file and save it again under the same name.

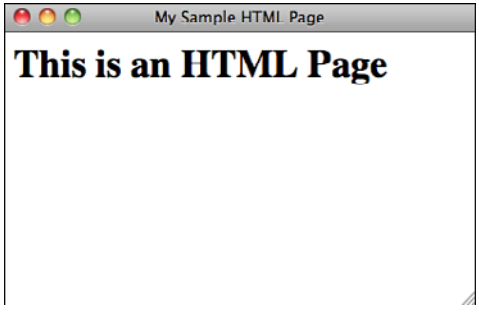


FIGURE 3.2
The sample HTML file.

Next, go back to your browser. Locate and choose a menu item or button called Refresh or Reload. The browser will read the new version of your file, and voilà! You can edit and preview and edit and preview until you get the file right.

▼ If you're getting the actual HTML text repeated in your browser rather than what's shown in Figure 3.2, make sure that your HTML file has an `.html` or `.htm` extension. This file extension tells your browser that it's an HTML file. The extension is important.

If things are going wrong—if you're getting a blank screen or you're getting some strange characters—something is wrong with your original file. If you've been using a word processor to edit your files, try opening your saved HTML file in a plain text editor. (Again, Notepad will work just fine.) If the text editor can't read the file or if the result is garbled, you haven't saved the original file in the right format. Go back into your original editor, and try saving the file as text only again. Then try viewing the file again in your browser until you get it right. ▲

Text Formatting and HTML

When an HTML page is parsed by a browser, any formatting you might have done by hand—that is, any extra spaces, tabs, returns, and so on—is ignored. The only thing that specifies formatting in an HTML page is an HTML tag. If you spend hours carefully editing a plain text file to have nicely formatted paragraphs and columns of numbers but don't include any tags, when a web browser loads the page, all the text will flow into one paragraph. All your work will have been in vain.

NOTE

There are two exceptions to this rule, a tag called `<pre>` and a CSS property. You'll learn about both of them in Lesson 7, "Formatting Text with HTML and CSS."

The advantage of having all whitespace (spaces, tabs, returns) ignored is that you can put your tags wherever you want. The following examples all produce the same output. Try them!

```
<h1>If music be the food of love, play on.</h1>
```

```
<h1>
If music be the food of love, play on.
</h1>
```

```
<h1>
If music be the food of love, play on.       </h1>
```

```
<h1>   If  music   be   the   food  of  love,
play   on. </h1 >
```

Using Cascading Style Sheets

Earlier in this lesson, I mentioned Cascading Style Sheets as a way you could control the look and feel of your pages. Styles are a way to control how the browser renders HTML tags (or elements, as they're called in standards documents). For example, in this lesson, I've used the `<h1>` tag a number of times. Most browsers print text enclosed inside an `<h1>` tag in a large, boldface font and leave some whitespace after the heading before printing something else. Using CSS, you can tell the browser to render the `<h1>` tag differently than it normally would. CSS provides a lot of flexibility in how you can alter the appearance of any type of element, and the styles can be applied in a number of different ways.

The advantage of CSS is that it can be used at varying levels of specificity. For example, you can put all your styles into a separate file and link to that file from your web page. That way, if you want to change the appearance of your site, you can simply edit your CSS file and make changes that span every page that links to your style sheet. Or if you prefer, you can include styles at the top of your page so that they apply only to that page. You can also include styles inside the tags themselves using the `style` attribute.

You can also control the specificity of the styles you create based on how you define them. For example, you can write rules that apply to all tags of a specific type, such as all `<h1>` elements. Or you can specify classes for your elements and then write rules that apply only to members of that class. Classes are categories or labels that are assigned to tags using the `class` attribute. For example, you could create a class called `headline` and then make all `<h1>` elements in the `headline` class red. You can also write rules that apply to single elements by assigning them a particular identifier using the `id` attribute and writing rules that apply to that identifier. Here's an example of an `<h1>` tag that includes both a class and an ID:

```
<h1 class="headline" id="leadstoryheadline">Lead Story Headline</h1>
```

One thing you'll find as you progress through the book is that CSS can serve as a replacement for many common tags. As I describe various tags, I explain how you can achieve the same effects using CSS instead. Generally, the flexibility of CSS means you should use HTML to describe the structure of pages and CSS to define their appearance. The coverage of CSS in this book culminates with Lesson 13, "Advanced CSS Techniques," which explains how to use CSS to manage the entire layout of the page, or even the entire layout of a site.

Including Styles in Tags

You've already seen how HTML pages are created using tags. I want to stop briefly and discuss attributes, as well. An attribute is an additional bit of information that somehow affects the behavior of a tag. Attributes are included inside the opening tag in a pair. Here's an example:

```
<tag attribute="value">
```

Some attributes can be used with nearly any tag; others are highly specific. One attribute that can be used with nearly any tag is `style`. By including the `style` attribute in a tag, you can include one or more style rules within a tag itself. Here's an example using the `<h1>` tag, which I introduced earlier:

```
<h1 style="font-family: Verdana, sans-serif;">Heading</h1>
```

The `style` attribute of the `<h1>` tag contains a style declaration. All style declarations follow this same basic pattern, with the property on the left and the value associated with that property on the right. The rule ends with a semicolon, and you can include more than one in a `style` attribute by placing commas between them. If you're only including one rule in the `style` attribute, the semicolon is optional, but it's a good idea to include it. In the preceding example, the property is `font-family`, and the value is `Verdana, sans-serif`. This attribute modifies the standard `<h1>` tag by changing the font to Verdana, and if the user doesn't have that font installed on his system, whichever sans-serif font the browser selects. (Sans-serif fonts are those that do not include *serifs*, the small lines at the ends of characters.)

Many, many properties can be used in style declarations. As previously mentioned, putting a declaration into a `style` attribute is just one of several ways that you can apply styles to your document.

Programs to Help You Write HTML

You might be thinking that all this tag stuff is a real pain, especially if you didn't get that small example right the first time. (Don't fret about it; I didn't get that example right the first time, and I created it.) You have to remember all the tags, and you have to type them in right and close each one. What a hassle!

Many freeware and shareware programs are available for editing HTML files. Most of these programs are essentially text editors with extra menu items or buttons that insert the appropriate HTML tags into your text. HTML-based text editors are particularly nice for two reasons: You don't have to remember all the tags, and you don't have to take the time to type them all. I've already mentioned HTML-Kit, but there are plenty of others.

Many general-purpose text editors also include special features to make it easier to deal with HTML files these days.

Many editors on the market purport to be WYSIWYG. These editors exchange ease of use (you don't have to remember the tags you use yourself) for control over your markup. To produce exactly the results you desire, editing the code yourself using a text editor is the best approach.

With that said, as long as you're aware that the result of working in those editors can vary, using WYSIWYG editors can be a quick way to create simple HTML pages. For professional web development and for using many of the advanced features, however, WYSIWYG editors can fall short, and you need to go under the hood to play with the HTML code anyhow. Even if you intend to use a WYSIWYG editor for the bulk of your HTML work, bear with me for the next couple of lessons and try these examples in text editors so that you get a feel for what HTML is before you decide to move on to an editor that hides the tags.

CAUTION

WYSIWYG editors tend to work best with files they've created themselves. If you have some existing HTML files that you need to edit, opening them in a WYSIWYG editor can do more harm than good, particularly if the files were created in a different WYSIWYG editor.

In addition to HTML and WYSIWYG editors, you can use converters, which take files from many popular word processing programs and convert them to HTML. With a simple set of templates, you can write your pages entirely in your favorite word processing program and then convert the result when you're done.

In many cases, converters can be extremely useful, particularly for putting existing documents on the Web as quickly as possible. However, converters suffer from many of the same problems as WYSIWYG editors. The results can vary from browser to browser, and many newer or advanced features aren't available in the converters. Also, most converter programs are fairly limited, not necessarily by their own features, but mostly by the limitations in HTML itself. No amount of fancy converting will make HTML do things that it can't do already. If a particular capability doesn't exist in HTML, the converter can't do anything to solve that problem. In fact, the converter might end up doing strange things to your HTML files, causing you more work than if you just did all the formatting yourself.

As previously mentioned, Appendix A lists many of the web page editors that are currently available. For now, if you have a simple HTML editor, feel free to use it for the examples in this book. If all you have is a text editor, no problem; you'll just have to do a little more typing.

Summary

In this lesson, you learned some basic points about what HTML is and how you define a text document as a web page. You learned a bit about the history of HTML and the reasons why the HTML specification has changed several times since the beginning. You also learned how CSS can be used to augment your HTML. You created your first web page with some basic tags. It wasn't so bad, was it? You also learned a bit about the current standard version of HTML and XHTML, and how to apply styles using CSS. In the following lesson, you expand on this and learn more about adding headings, text, and lists to your pages.

Workshop

Now that you've had an introduction to HTML and a taste of creating your first (simple) web page, here's a workshop that will guide you toward more of what you'll learn. A couple of questions and answers that relate to HTML formatting are followed by a brief quiz and answers about HTML. The exercises prompt you to examine the code of a more advanced page in your browser.

Q&A

Q Can I do *any* formatting of text in HTML?

A You can do some formatting to strings of characters. CSS has superseded most of the tags for formatting text, and nearly all of them have been removed from HTML5; however, browsers still support the older text formatting elements, and a few formatting tags do still remain. You'll learn some formatting tricks in Lesson 7.

Q I'm using Windows. My word processor won't let me save a text file with an extension that's anything except `.txt`. If I type in `index.html`, my word processor saves the file as `index.html.txt`. What can I do?

A You can rename your files after you've saved them so that they have an `.html` or `.htm` extension, but having to do so can be annoying if you have a large number of files. Consider using a text editor or HTML editor for your web pages.

Quiz

1. What does HTML stand for? How about XHTML?
2. What's the primary function of HTML?
3. Why doesn't HTML control the layout of a page?
4. What's the basic structure of an HTML tag?

Quiz Answers

1. HTML stands for Hypertext Markup Language. XHTML stands for Extensible Hypertext Markup Language.
2. HTML enables you to describe the structure of a document so that it can be styled, either using HTML tags or using CSS.
3. HTML doesn't control the layout of a page, because it's designed to be cross-platform. It takes the differences of many platforms into account and allows all browsers and all computer systems to be on equal ground.
4. Most HTML elements consist of opening and closing tags, and they surround the text that they affect. The tags are enclosed in brackets (<>). The beginning tag turns on a feature, and the ending tag, which is preceded by a forward slash (/), turns it off.

Exercises

1. Before you actually start writing a meatier HTML page, getting a feel for what an HTML page looks like certainly helps. Luckily, you can find plenty of source material to look at. Every page that comes over the wire to your browser is in HTML (or perhaps XHTML) format. (You almost never see the codes in your browser; all you see is the final result.)

   Most web browsers have a way of letting you see the HTML source of a web page. If you're using Internet Explorer, for example, navigate to the web page that you want to look at. Choose View, Source to display the source code in a text window. In Firefox, choose View, Page Source.

   Try going to a typical home page and then viewing its source. For example, Figure 3.3 shows the home page for Craigslist, a free online classified ads service search page at http://www.craisglist.org/.

FIGURE 3.3
Craigslist home page.

The HTML source code looks something like Figure 3.4.

Source of: http://newyork.craigslist.org/
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>craigslist: new york city classifieds for jobs, apartments, personals, for sale, services, comm
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> <meta name="description" cont
<meta name="keywords" content="new york city, classifieds, craigslist, craig s list, craiglist, jobs,
<meta name="verify-v1" content="rv2467y3Du3pMljc9GUaMWWVQ2pnfofViCaVr7zmqQ8=">
<link rel="stylesheet" title="craigslist" href="http://www.craigslist.org/styles/craigslist.css" type=
<!--[if lt IE 6]>
<link rel="stylesheet" title="craigslist" href="http://www.craigslist.org/styles/cl-ie5.css"
<![endif]-->
<link rel="shortcut icon" href="http://www.craigslist.org/favicon.ico" type="image/x-icon">
</head>
<body class="hp">
<table cellspacing="0" cellpadding="0" summary="page">
<tr>
<td id="leftbar">

craigslist

<ul id="postlks">
<a href="https://post.craigslist.org/nyc">post to classifi
my account
<a href="http://www.craigslist.org/about/help/">help, faq,

<form id="search" action="/search/" method="GET">
search craigslist
<input type="hidden" name="areaID" value="3">
<input type="hidden" name="subAreaID" value="">
<input type="hidden" name="query" value="">
<input type="hidden" name="catAbbreviation" value="sss">
<input type="hidden" name="minAsk" value="">
<input type="hidden" name="maxAsk" value="">
<input type="hidden" name="hasPic" value="">
<input type="hidden" name="zoomToPosting" value="">
<input type="hidden" name="query" value="">
<input type="hidden" name="srchType" value="A">
<input type="hidden" name="minAsk" value="">
<input type="hidden" name="maxAsk" value="">
<input type="hidden" name="bedrooms" value="">
<input type="hidden" name="housing_type" value="">
<input type="hidden" name="addTwo" value="purrr

LESSON 4

Learning the Basics of HTML

In the first three lessons, you learned about the World Wide Web, how to organize and plan your websites, and why you need to use HTML to create a web page. In Lesson 3, "Introducing HTML and XHTML," you even created your first simple web page. In this lesson, you learn about each of the basic HTML tags in more depth and begin writing web pages with headings, paragraphs, and several different types of lists. We focus on the following topics and HTML tags:

- Tags for overall page structure: `<html>`, `<head>`, and `<body>`
- Tags for titles, headings, and paragraphs: `<title>`, `<h1>` through `<h6>`, and `<p>`
- Tags for comments: `<!--...-->`

Structuring Your HTML

HTML defines three tags that describe the page's overall structure and provide some simple header information. These three tags—<html>, <head>, and <body>—make up the basic skeleton of every web page. They also provide simple information about the page (such as its title or its author) before loading the entire thing. The page structure tags don't affect what the page looks like when it displays; they're only there to help tools that interpret or filter HTML files.

At one time, these tags were optional. Even today, if you leave them out of your markup, browsers usually can read the page anyway. These tags, however, *are* required elements in HTML 4/XHTML 1.1 and HTML5, the standards supported by current browsers.

The DOCTYPE Identifier

Although it's not a page structure tag, the XHTML 1.0 and HTML5 standards include an additional requirement for your web pages. The first line of each page must include a DOCTYPE identifier that defines the XHTML 1.0 version to which your page conforms, and the Document Type Definition (DTD) that defines the specification. This is followed by the <html>, <head>, and <body> tags. In the following example, the HTML5 document type appears before the page structure tags:

```
<!DOCTYPE html><html>
<head>
<title>Page Title</title>
</head>
<body>
...your page content...
</body>
</html>
```

Refer to Lesson 18, "Writing Good Web Pages: Do's and Don'ts," for more information about the DOCTYPE tag, and more information about the differences between the various document types.

The <html> Tag

The first page structure tag in every HTML page is the <html> tag. It indicates that the content of this file is in the HTML language. In the XHTML 1.0 recommendation, the <html> tag should follow the DOCTYPE identifier (as mentioned in the previous note) as shown in the following example.

All the text and HTML elements in your web page should be placed within the beginning and ending HTML tags, like this:

```
<!DOCTYPE html><html>
...your page...
</html>
```

The <html> tag serves as a container for all the tags that make up the page. It is required because both XML and SGML specify that every document have a root element. Were you to leave it out, which you shouldn't do because it would make your page invalid, the browser would make up an <html> tag for you so that the page would make sense to its HTML processor.

The <head> Tag

The <head> tag is a container for the tags that contain information about the page, rather than information that will be displayed on the page. Generally, only a few tags are used in the <head> portion of the page (most notably, the page title, described later). You should never put any of the text of your page into the header (between <head> tags).

Here's a typical example of how you properly use the <head> tag. (You'll learn about <title> later.)

```
<!DOCTYPE html><html>
<head>
<title>This is the Title. It will be explained later on</title>
</head>
...your page...
</html>
```


The <body> Tag

The content of your HTML page (represented in the following example as *...your page...*) resides within the <body> tag. This includes all the text and other content (links, pictures, and so on). In combination with the <html> and <head> tags, your page will look something like this:

```
<!DOCTYPE html><html>
<head>
<title>This is the Title. It will be explained later on</title>
</head>
<body>
...your page...
</body>
</html>
```

You might notice here that the tags are nested. That is, both <body> and </body> tags go inside both <html> tags; the same with both <head> tags. All HTML tags work this way, forming individual nested sections of text. You should be careful never to overlap tags. That is, never do something like the following:

```
<!DOCTYPE html><html>
<head>
<body>
</head>
</body>
</html>
```

Whenever you close an HTML tag, make sure that you close the most recent unclosed tag. (You learn more about closing tags as you go on.)

NOTE

In HTML, closing some tags is optional. In HTML 4.0 and earlier, closing tags were forbidden in some cases. HTML standards that require your markup to be well-formed XML (like XHTML 1.0) require that all tags be closed. If you're just learning HTML, this won't be a big deal. If you already have a passing familiarity with the language, however, this might surprise you. The examples shown in this book display the proper way to close tags so that older browsers will interpret XHTML 1.0 closures correctly.

The Title

Each HTML page needs a title to indicate what the page describes. It appears in the title bar of the browser when people view the web page. The title is stored in your browser's favorites (or bookmarks) and also in search engines when they index your pages. Use the <title> tag to give a page a title.

<title> tags are placed within the <head> tag and normally describe the contents of the page, as follows:

```
<!DOCTYPE html><html>
<head>
<title>The Lion, The Witch, and the Wardrobe</title>
</head>
<body>
...your page...
</body>
</html>
```

Each page can have only one title, and that title can contain only plain text; that is, no other tags should appear inside the title.

Try to choose a title that's both short and descriptive of the content. Your title should be relevant even out of context. If someone browsing on the Web follows a random link and ends up on this page, or if a person finds your title in a friend's browser history list, would he have any idea what this page is about? You might not intend the page to be used independently of the pages you specifically linked to it, but because anyone can link to any page at any time, be prepared for that consequence and pick a helpful title.

NOTE

When search engines index your pages, each page title is captured and listed in the search results. The more descriptive your page title, the more likely it is that someone will choose your page from all the search results.

Also, because browsers put the title in the title bar of the window, you might have a limited amount of space. (Although the text within the `<title>` tag can be of any length, it might be cut off by the browser when it displays.) Here are some other examples of good titles:


```
<title>Poisonous Plants of North America</title>
<title>Image Editing: A Tutorial</title>
<title>Upcoming Cemetery Tours, Summer 1999</title>
<title>Installing the Software: Opening the CD Case</title>
<title>Laura Lemay's Awesome Home Page</title>
```

Here are some not-so-good titles:

```
<title>Part Two</title>
<title>An Example</title>
<title>Nigel Franklin Hobbes</title>
<title>Minutes of the Second Meeting of the Fourth Conference of the
Committee for the Preservation of English Roses, Day Four, After Lunch</title>
```

Figure 4.1 shows how the following title looks in a browser:

```
<title>Poisonous Plants of North America</title>
```

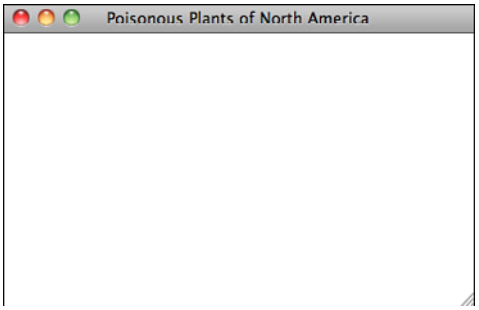


FIGURE 4.1
A page containing only header elements.

Headings

Headings add titles to sections of a page. HTML defines six levels of headings. Heading tags look like the following:

```
<h1>Installing Your Safetee Lock</h1>
```

The numbers indicate heading levels (`h1` through `h6`). The headings, when they display, aren't numbered. They display in larger or bolder text, centered or underlined, or capitalized—so that they stand out from regular text.

Think of the headings as items in an outline. If the text you write is structured, use the headings to express that structure, as shown in the following code:

```
<h1>Movies</h1>
  <h2>Action/Adventure</h2>
     <h3>Caper</h3>
     <h3>Sports</h3>
     <h3>Thriller</h3>
     <h3>War</h3>
  <h2>Comedy</h2>
     <h3>Romantic Comedy</h3>
     <h3>Slapstick</h3>
  <h2>Drama</h2>
     <h3>Buddy Movies</h3>
     <h3>Mystery</h3>
     <h3>Romance</h3>
  <h2>Horror</h2>
```

Notice that I indented the headings in this example to better show the hierarchy. They don't have to be indented in your page; in fact, the browser ignores the indenting.

TIP

Even though the browser ignores any indenting you include in your code, you will probably find it useful to indent your code so that it's easier to read. You'll find that any lengthy examples in this book are indented for that reason, and you'll probably want to carry that convention over to your own HTML code.

Unlike titles, headings can be any length, spanning many lines of text. Because headings are emphasized, however, having many lines of emphasized text might be tiring to read.

A common practice is to use a first-level heading at the top of your page to either duplicate the title (which usually is displayed elsewhere), or to provide a shorter or less context-specific form of the title. If you have a page that shows several examples of folding bed sheets—for example, part of a long presentation on how to fold bed sheets—the title might look something like the following:

```
<title>How to Fold Sheets: Some Examples</title>
```

The topmost heading, however, might just be as follows:

```
<h1>Examples</h1>
```


CAUTION

Don't use headings to display text in boldface type or to make certain parts of your page stand out more. Although the result might look cool in your browser, you don't know what it'll look like when other people use their browsers to read your page. Other browsers might number headings or format them in a manner that you don't expect.

Tools to create searchable indexes of web pages might extract your headings to indicate the important parts of a page. By using headings for something other than an actual heading, you might be foiling those search programs and creating strange results.

Figure 4.2 shows the following headings as they appear in a browser:

Input ▼

```
<h1>Mythology Through the Ages</h1>
  <h2>Common Mythological Themes</h2>
  <h2>Earliest Known Myths</h2>
  <h2>Origins of Mythology</h2>
   <h3>Mesopotamian Mythology</h3>
   <h3>Egyptian Mythology</h3>
     <h4>The Story of Isis and Osiris</h4>
     <h4>Horus and Set: The Battle of Good vs. Evil</h4>
     <h4>The Twelve Hours of the Underworld</h4>
     <h4>The River Styx</h4>
  <h2>History in Myth</h2>
```

Output ▼

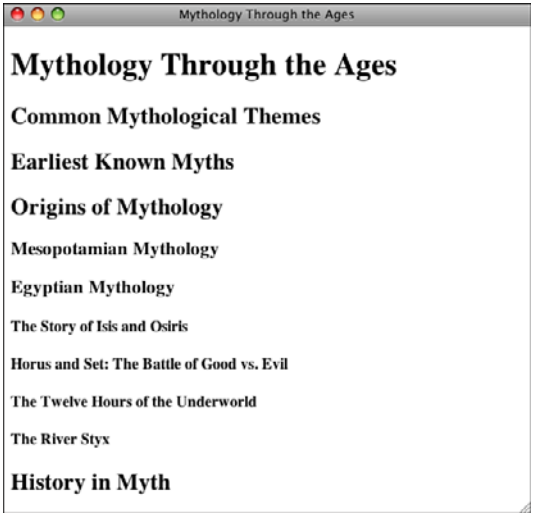


FIGURE 4.2
HTML heading elements.

TIP

From a visual perspective, headings 4 through 6 aren't visually interesting, but they do have meaning in terms of the document's structure. If using more than three levels of headings makes sense for the document you're creating, you can use those tags and then use CSS to make them appear as you intend.

Paragraphs

Now that you have a page title and several headings, you can add some ordinary paragraphs to the page.

Paragraphs are created using the <p> tag. The Enigern story should look like this:

```
<p>Slowly and deliberately, Enigern approached the mighty dragon.
A rustle in the trees of the nearby forest distracted his attention
for a brief moment, a near fatal mistake for the brave knight.</p>
<p>The dragon lunged at him, searing Enigern's armor with a rapid
blast of fiery breath. Enigern fell to the ground as the dragon
hovered over him. He quickly drew his sword and thrust it into the
dragon's chest.</p>
```

What if you want more (or less) space between your paragraphs than the browser provides by default? The answer is to use CSS. As you'll see, it provides fine control over the spacing of elements on the page, among other things. Figure 4.3 shows what happens when I add another paragraph about Enigern and the dragon to the page. The paragraph breaks are added between the closing and opening <p> tags in the text.

Input ▼

```
<p>The dragon fell to the ground, releasing an anguished cry and
seething in pain. The thrust of Enigern's sword proved fatal as
the dragon breathed its last breath. Now Enigern was free to
release Lady Aelfleada from her imprisonment in the dragon's lair. </p>
```

Output ▼

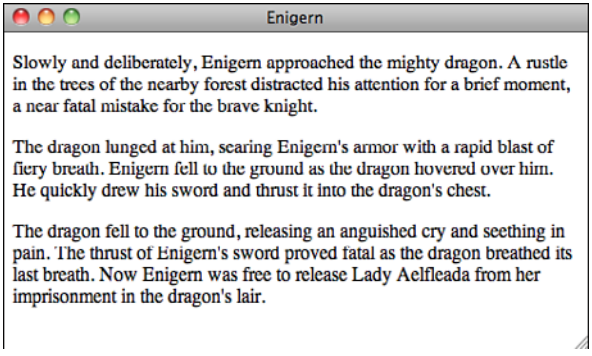


FIGURE 4.3
An HTML paragraph.

Comments

You can put comments into HTML pages to describe the page itself or to provide some kind of indication of the status of the page. Some source code control programs store the page status in comments, for example. Text in comments is ignored when the HTML file

is parsed; comments don't ever show up onscreen—that's why they're comments. Comments look like the following:

```
<!-- This is a comment -->
```

Here are some examples:

```
<!-- Rewrite this section with less humor -->
<!-- Neil helped with this section -->
<!-- Go Tigers! -->
```

As you can see from Figure 4.4, users can view your comments using the View Source functionality in their browsers, so don't put anything in comments that you don't want them to see.

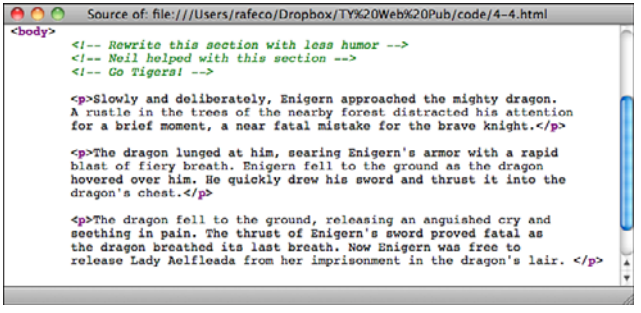


FIGURE 4.4
HTML comments displayed within the source for a page.

Task: Exercise 4.1: Creating a Real HTML Page

At this point, you know enough to start creating simple HTML pages. You understand what HTML is, you've been introduced to a handful of tags, and you've even opened an HTML file in your browser. You haven't created any links yet, but you'll get to that soon enough, in Lesson 8, "Using CSS to Style a Site."

This exercise shows you how to create an HTML file that uses the tags you learned about up to this point. It gives you a feel for what the tags look like when they display onscreen and for the sorts of typical mistakes you're going to make. (Everyone makes them, and that's why using an HTML editor that does the typing for you is often helpful. The editor doesn't forget the closing tags, leave off the slash, or misspell the tag.)

So, create a simple example in your text editor. Your example doesn't have to say much of anything; all it needs to include are the structure tags, a title, a couple of headings, and a paragraph or two. Here's an example:

Input

```
<!DOCTYPE html><html>
<head>
<title>Camembert Incorporated</title>
</head>
<body>
<h1>Camembert Incorporated</h1>
<p>"Many's the long night I dreamed of cheese -- toasted, mostly."
-- Robert Louis Stevenson</p>
<h2>What We Do</h2>
<p>We make cheese. Lots of cheese; more than eight tons of cheese
a year.</p>
<h2>Why We Do It</h2>
<p>We are paid an awful lot of money by people who like cheese.
So we make more.</p>
</body>
</html>
```

Save the example to an HTML file, open it in your browser, and see how it came out.

If you have access to a device other than a computer that has access to the Web, like a mobile phone, check out your page there as well, and take note of the differences between them. Figure 4.5 shows what the cheese factory example looks like.

Output

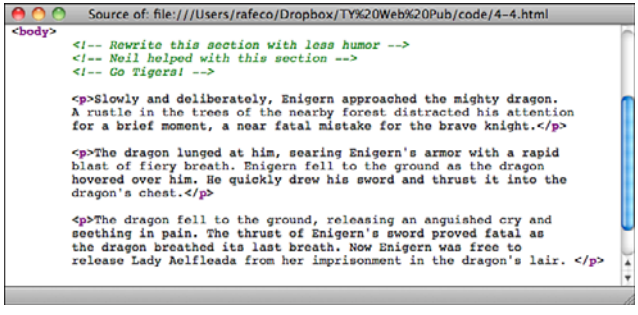


FIGURE 4.5
The cheese factory example.

Summary

HTML, a text-only markup language used to describe hypertext pages on the World Wide Web, describes the structure of a page, not its appearance.

In this lesson, you learned what HTML is and how to write and preview simple HTML files. You also learned about the HTML tags shown in Table 4.1.

TABLE 4.1 HTML Tags from Lesson 4

Tag	Attribute	Use
`<html> .. </html>`		The entire HTML page
`<head> .. </head>`		The head, or prologue, of the HTML page
`<body> .. </body>`		All the other content in the HTML page
`<title> .. </title>`		The title of the page
`<h1> .. </h1>`		First-level heading
`<h2> .. </h2>`		Second-level heading
`<h3> .. </h3>`		Third-level heading
`<h4> .. </h4>`		Fourth-level heading
`<h5> .. </h5>`		Fifth-level heading
`<h6> .. </h6>`		Sixth-level heading
`<p> .. </p>`		A paragraph

Workshop

You've learned a lot in this lesson, and the following workshop can help you remember some of the most important points. I've anticipated some of the questions you might have in the first section of the workshop.

Q&A

Q In some web pages, I've noticed that the page structure tags (`<html>`, `<head>`, `<body>`) aren't used. Do I really need to include them if pages work just fine without them?

A Most browsers handle plain HTML without the page structure tags. The XHTML 1.0 recommendation requires that these tags appear in your pages. It's a good idea to get into the habit of using them now. Including the tags allows your pages to be read by more general SGML tools and to take advantage of features of future browsers. And using these tags is the correct thing to do if you want your pages to conform to true HTML format.

Q Is the tag the general-purpose tag for use when styling a page?

A No. The tag is the general purpose tag for containing content on a page. The tag is intended specifically to hold paragraphs of text. There are many tags that are not valid when placed within a tag, including . You learn more about in Lesson 7, "Formatting Text with HTML and CSS."

Q Is it possible to put HTML tags inside comments?

A Yes, you can enclose HTML tags within comments, and the browser will not display them. It's common to use comments to temporarily hide sections of a page, especially when testing things. Programmers (and web developers) generally refer to this as "commenting it out."

Quiz

1. What three HTML tags describe the overall structure of a web page, and what do each of them define?
2. Where does the <title> tag go, and what is it used for?
3. How many different levels of headings does HTML support? What are their tags?
4. Why is it a good idea to use two-sided paragraph tags, even though the closing tag is optional in HTML?

Quiz Answers

1. The <html> tag indicates that the file is in the HTML language. The <head> tag specifies that the lines within the beginning and ending points of the tag are the prologue to the rest of the file. The <body> tag encloses the remainder of your HTML page (text, links, pictures, and so on).
2. The <title> tag indicates the title of a web page in a browser's bookmarks, hotlist program, or other programs that catalog web pages. This tag always goes inside the <head> tags.
3. HTML supports six levels of headings. Their tags are <h1 .. /h1> through <h6 .. /h6>.
4. The closing tag becomes important when aligning text to the left, right, or center of a page. (Text alignment is discussed in Lesson 7.) Closing tags also are required for XHTML 1.0.

Exercises

1. Using the Camembert Incorporated page as an example, create a page that briefly describes topics that you would like to cover on your own website. You'll use this page to learn how to create your own links in Lesson 5.
2. Create a second page that provides further information about one of the topics you listed in the first exercise. Include a couple of subheadings (such as those shown in Figure 4.2). If you feel adventurous, complete the page's content and include lists where you think they enhance the page. This exercise can also help prepare you for Lesson 5.

LESSON 5

Organizing Information with Lists

In the previous lesson, you learned about the basic elements that make up a web page. This lesson introduces lists, which, unlike the other tags that have been discussed thus far, are composed of multiple tags that work together. As you'll see, lists come in a variety of types, and can be used not only for traditional purposes, like shopping lists or bulleted lists, but also for creating outlines or even navigation for websites. In this lesson, you'll learn the following:

- How to create numbered lists
- How to create bulleted lists
- How to create definition lists
- The *Cascading Style Sheets (CSS)* properties associated with lists

Lists: An Overview

Lists are a general-purpose container for collections of things. They come in three varieties. Ordered lists are numbered and are useful for presenting things like your top 10 favorite songs from 2010 or the steps to bake a cake. Unordered lists are not numbered and by default are presented with bullets for each list item. However, these days unordered lists are often used as a general-purpose container for any list-like collection of items. Yes, they're frequently used for bulleted lists of the kind you might see on a PowerPoint slide, but they're also used for things like collections of navigation links and even pull-down menus. Finally, definition lists are used for glossaries and other items that pair a label with some kind of description.

NOTE

Older HTML standards also supported two additional list types: menu lists (`<menu>`) and directory lists (`<dir>`). Menu lists were deprecated until HTML5, but they have been reinstated for use as lists of commands.

All the list tags have the following common elements:

- Each list has an outer element specific to that type of list. For example, `<ul>` and `</ul>` for unordered lists or `<ol>` and `</ol>` for ordered lists.
- Each list item has its own tag: `<dt>` and `<dd>` for the glossary lists and `<li>` for the other lists.

NOTE

The closing tags for `<dd>`, `<dt>`, and `<li>` were optional in HTML. To comply with HTML5, use closing tags `</dd>`, `</dt>`, and `</li>`.

Although the tags and the list items can be formatted any way you like in your HTML code, I prefer to arrange the tags so that the list tags are on their own lines and each new item starts on a new line. This way, you can easily select the whole list and the individual elements. In other words, I find the following HTML

```
<p>Dante's Divine Comedy consists of three books:</p>
<ul>
  <li>The Inferno</li>
  <li>The Purgatorio</li>
  <li>The Paradiso</li>
</ul>
```

is easier to read than

```
<p>Dante's Divine Comedy consists of three books:</p>
<ul><li>The Inferno</li><li>The Purgatorio</li><li>The Paradiso</li></ul>
```

although both result in the same output in the browser.

Numbered Lists

Numbered lists are surrounded by the `<ol>...</ol>` tags (`ol` stands for *ordered list*), and each item within the list is included in the `<li>...</li>` (list item) tag. When the browser displays an ordered list, it numbers and indents each of the elements sequentially. You don't have to perform the numbering yourself and, if you add or delete items, the browser renumbers them the next time the page loads.

Ordered lists are lists in which each item is numbered or labeled with a counter of some kind (like letters or roman numerals). Use numbered lists only when the sequence of items on the list is relevant. Ordered lists are good for steps to follow or instructions to the readers, or when you want to rank the items in a list. If you just want to indicate that something has a number of elements that can appear in any order, use an unordered list instead.

For example, the following is an ordered list of steps that explain how to install a new operating system. You can see how the list displays in a browser in Figure 5.1.

Input ▼

```
<h1>Installing Your New Operating System</h1>
<ol>
<li>Insert the CD-ROM into your CD-ROM drive.</li>
<li>Choose RUN.</li>
<li>Enter the drive letter of your CD-ROM (example: D:\),
followed by SETUP.EXE.</li>
<li>Follow the prompts in the setup program.</li>
<li>Reboot your computer after all files are installed.</li>
<li>Cross your fingers.</li>
</ol>
```

Output ▶

FIGURE 5.1
An ordered list in HTML.

Customizing Ordered Lists

Attributes are a structural element of HTML that haven't been introduced yet. They provide further information about a tag or modify the behavior of a tag and take the form of an attribute name and, optionally, an attribute value. For example, all tags support the `style` attribute. Here's what an attribute for an HTML element looks like:

```
<ul style="insert style information here">
```

Since Cascading Style Sheets were introduced, HTML elements and attributes that describe how the page should be displayed have been migrating to CSS. HTML is now associated with describing the structure of a page's content—CSS is for indicating exactly how the page should look in the browser. In discussing how to customize lists, I talk about the old HTML attributes that were once used (and still work in nearly all browsers), and the more current CSS approach to solving the same problems.

The display attributes for ordered lists enable you to control several features of the lists including which numbering scheme to use and from which number to start counting (if you don't want to start at 1). The attributes mentioned in this section are deprecated in favor of using style sheet properties and that can accomplish the same task. You may, however, see these attributes used in examples or in other people's markup.

You can customize ordered lists in two main ways: the numbering style and the number with which the list starts. The `type` attribute can set the numbering style, or you can use the `list-style-type` CSS property. Table 5.1 lists the numbering styles.

TABLE 5.1 Ordered List Numbering Styles

CSS Value	Attribute Value	Description
`decimal`	1	Standard Arabic numerals (1, 2, 3, 4, and so on)
`lower-alpha`	a	Lowercase letters (a, b, c, d, and so on)
`upper-alpha`	A	Uppercase letters (A, B, C, D, and so on)
`lower-roman`	i	Lowercase Roman numerals (i, ii, iii, iv, and so on)
`upper-roman`	I	Uppercase Roman numerals (that is, I, II, III, IV, and so on)

You can specify types of numbering in the `<ol>` tag as follows: `<ol type="a">` or using the style attribute, like this:

```
<ol style="list-style-type: lower-alpha">
```

By default, the `decimal` type is assumed.

NOTE

HTML5 has dropped support for the `type` property, and all but the oldest browsers support `list-style-type`, so you should take that approach. However, you may run into old Web pages or tools for creating Web pages that still use the type `attribute`.

As an example, consider the following list:

```
<p>The Days of the Week in French:</p>
<ol>
<li>Lundi</li>
<li>Mardi</li>
<li>Mercredi</li>
<li>Jeudi</li>
<li>Vendredi</li>
<li>Samedi</li>
<li>Dimanche</li>
</ol>
```

If you were to set the list style type `upper-roman` to the `<ol>` tag, as follows, it would appear in a browser, as shown in Figure 5.2:

Input ▼

```
<h1>The days of the week in French</h1>
<ol style="list-style-type: upper-roman">
<li>Lundi</li>
<li>Mardi</li>
<li>Mercredi</li>
<li>Jeudi</li>
<li>Vendredi</li>
<li>Samedi</li>
<li>Dimanche</li>
</ol>
```

Output ▶

FIGURE 5.2
An ordered list displayed using an alternative numbering style.

You can also use the `list-style-type` property or the `type` attribute with the `<li>` tag, changing the numbering type in the middle of the list. When the numbering style is changed in an `<li>` tag, it affects that item and all entries following it in the list. Using the `start` attribute, you can specify the number or letter with which to start your list. The default starting point is 1, of course. You can change this number by using `start`. `<ol start="4">`, for example, would start the list at number 4, whereas `<ol type="a" start="3">` would start the numbering with `c` and move through the alphabet from there. The value for the `start` attribute should always be a decimal number, regardless of the numbering style being used.

For example, you can list the last 6 months of the year and start numbering with the Roman numeral `VII`. The results appear in Figure 5.3.

Input ▼

```
<p>The Last Six Months of the Year (and the Beginning of the Next Year):</p>
<ol style="list-style-type: upper-roman" start="7">
<li>July</li>
<li>August</li>
<li>September</li>
<li>October</li>
<li>November</li>
```

```
<li>December</li>
<li style="list-style-type: lower-roman">January</li>
</ol>
```

Output ▶

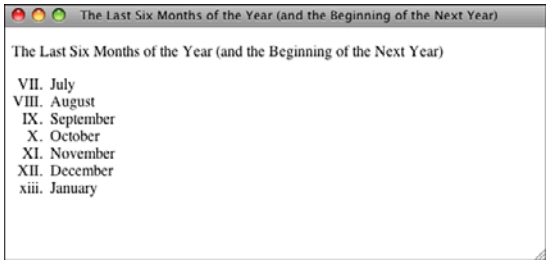


FIGURE 5.3
An ordered list with an alternative numbering style and starting number.

As with the `type` attribute, you can change the value of an entry's number at any point in a list. You do so by using the `value` attribute in the `<li>` tag. Assigning a `value` in an `<li>` tag restarts numbering in the list starting with the affected entry.

Suppose that you wanted the last three items in a list of ingredients to be `10`, `11`, and `12` rather than `6`, `7`, and `8`. You can reset the numbering at `Eggs` using the `value` attribute, as follows:

```
<h1>Cheesecake Ingredients</h1>
<ol>
<li>Quark Cheese</li>
<li>Honey</li>
<li>Cocoa</li>
<li>Vanilla Extract</li>
<li>Flour</li>
<li value="10">Eggs</li>
<li>Walnuts</li>
<li>Margarine</li>
</ol>
```


Unordered Lists

Unordered lists are often referred to as bulleted lists. Instead of being numbered, each element in the list has the same marker. The markup to create an unordered list looks just like an ordered list except that the list is created by using `<ul>...</ul>` tags rather than `ol`. The elements of the list are placed within `<li>` tags, just as with ordered lists.

Browsers usually mark each item in unordered lists using bullets or some other symbol; Lynx, a text browser, inserts an asterisk (`*`).

The following input and output example shows an unordered list. Figure 5.4 shows the results in a browser.

Input ▼

```
<p>Things I like to do in the morning:</p>
<ul>
<li>Drink a cup of coffee</li>
<li>Watch the sunrise</li>
<li>Listen to the birds sing</li>
<li>Hear the wind rustling through the trees</li>
<li>Curse the construction noises for spoiling the peaceful mood</li>
</ul>
```

Output ▶

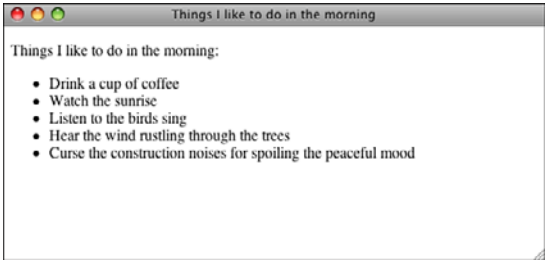


FIGURE 5.4
An unordered list.

Customizing Unordered Lists

As with ordered lists, unordered lists can be customized using the `type` attribute or the `list-style-type` property. By default, most browser use solid bullets to mark entries in unordered lists. Text browsers such as Lynx generally opt for an asterisk. The other bullet styles are as follows:

- **`"disc"`**—A disc or bullet; this style is the default.
- **`"square"`**—Obviously, a square rather than a disc.
- **`"circle"`**—As compared with the `disc`, which most browsers render as a filled circle, this value should generate an unfilled circle.

In this case, the values for `list-style-type` and for the `type` attribute are the same. In the following input and output example, you see a comparison of these three types as rendered in a browser (see Figure 5.5):

Input ▼

```
<ul style="list-style-type: disc">
  <li>DAT - Digital Audio Tapes</li>
  <li>CD - Compact Discs</li>
  <li>Cassettes</li>
</ul>
<ul style="list-style-type: square">
  <li>DAT - Digital Audio Tapes</li>
  <li>CD - Compact Discs</li>
  <li>Cassettes</li>
</ul>
<ul style="list-style-type: circle">
  <li>DAT - Digital Audio Tapes</li>
  <li>CD - Compact Discs</li>
  <li>Cassettes</li>
</ul>
```

Output ▶

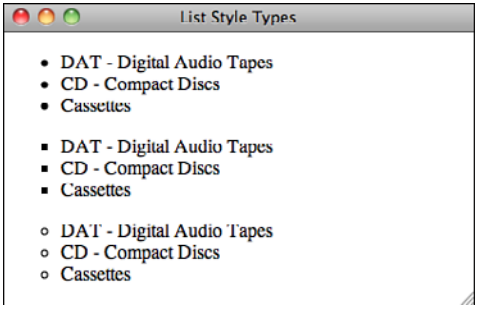


FIGURE 5.5
Unordered lists with different bullet types.

If you don't like any of the bullet styles used in unordered lists, you can substitute an image of your own choosing in place of them. To do so, use the `list-style-image` property. By setting this property, you can use an image of your choosing for the bullets in your list. Here's an example:

```
<ul style="list-style-image: url(/bullet.gif)">
    <li>Example</li>
</ul>
```

Don't worry much about what this all means right now. Images are discussed in Lesson 9, "Adding Images, Color, and Backgrounds." Right now, all you need to know is that the URL in parentheses should point to the image you want to use.

As you've seen in the screenshots so far, when items are formatted in a list and the list item spans more than one line, the lines of text that follow the first are aligned with the

beginning of the text on the first line. If you prefer that they begin at the position of the bullet or list number, as shown in Figure 5.6, use the `list-style-position` property:

```
<ul style="list-style-position: inside;">
        <li>Always use Pillsbury's Best Flour.</li>
        <li>Sift flour twice before adding to cakes or breakfast cakes.</li>
        <li>Make all measurements level by using edge of knife to lightly scrape
off from top of cup or spoon until material is even with the edges.</li>
        <li>Use same sized cups or spoons in measuring for the same recipe.</li>
        <li>Before starting to make recipe, read through carefully, then put on
table all the materials and tools needed in making that particular recipe.</li>
</ul>
```

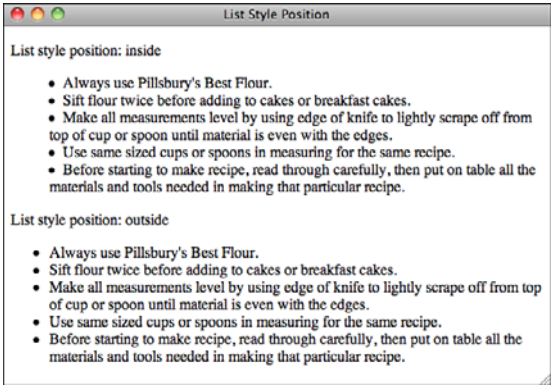


FIGURE 5.6
How the `list-style-position` property affects the layout of lists.

The default value is `outside`, and the only alternative is `inside`. Finally, if you want to modify several list-related properties at once, you can simply use the `list-style` property. You can specify three values for `list-style`: the list style type, the list style position, and the URL of the image to be used as the bullet style. This property is just a shortcut for use if you want to manipulate several of the list-related properties simultaneously. Here's an example:

```
<ul style="list-style: circle inside URL(/bullet.gif)">
    <li>Example</li>
</ul>
```

Glossary Lists

Glossary lists are slightly different from other lists. Each list item in a glossary list has two parts:

- A term
- The term's definition

Each part of the glossary list has its own tag: `<dt>` for the term (*definition term*) and `<dd>` for its definition (*definition definition*). `<dt>` and `<dd>` usually occur in pairs, although most browsers can handle single terms or definitions. The entire glossary list is indicated by the tags `<dl>...</dl>` (*definition list*).

The following is a glossary list example with a set of herbs and descriptions of how they grow (see Figure 5.7):

Input ▼

```
<dl>
<dt>Basil</dt>
<dd>Annual. Can grow four feet high; the scent of its tiny white
flowers is heavenly</dd>
<dt>Oregano</dt>
<dd>Perennial. Sends out underground runners and is difficult
to get rid of once established.</dd>
<dt>Coriander</dt>
<dd>Annual. Also called cilantro, coriander likes cooler
weather of spring and fall.</dd>
</dl>
```

Output ▶

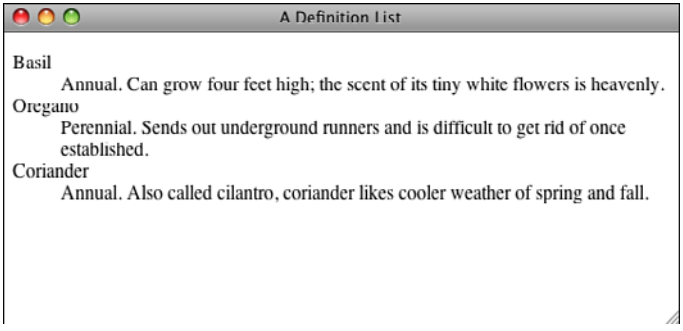


FIGURE 5.7
A glossary list.

Glossary lists usually are formatted in browsers with the terms and definitions on separate lines, and the left margins of the definitions are indented.

You don't have to use glossary lists for terms and definitions, of course. You can use them anywhere that the same sort of list is needed. Here's an example involving a list of frequently asked questions:

```
<dl>
<dt>
<dt>What is the WHATWG?</dt>
<dd>The Web Hypertext Application Technology Working Group (WHATWG) is a growing
```

```
community of people interested in evolving the Web. It focuses primarily on the
development of HTML and APIs needed for Web applications.</dd>
<dt>What is the WHATWG working on?</dt>
<dd>The WHATWG's main focus is HTML5. The WHATWG also works on Web Workers and
occasionally specifications outside WHATWG space are discussed on the WHATWG
mailing list and forwarded when appropriate.</dd>
<dt>How can I get involved?</dt>
<dd>There are lots of ways you can get involved, take a look and see What you
can do!</dd>
<dt>Is participation free?</dt>
<dd>Yes, everyone can contribute. There are no memberships fees involved, it's
an open process. You may easily subscribe to the WHATWG mailing lists. You may
also join the the W3C's new HTMLWG by going through the slightly longer applica-
tion process.</dd>
</dl>
```

Nesting Lists

What happens if you put a list inside another list? Nesting lists is fine as far as HTML is concerned; just put the entire list structure inside another list as one of its elements. The nested list just becomes another element of the first list, and it's indented from the rest of the list. Lists like this work especially well for menu-like entities in which you want to show hierarchy (for example, in tables of contents) or as outlines.

Indenting nested lists in HTML code itself helps show their relationship to the final layout:

```
<ol>
        <li>WWW</li>
        <li>Organization</li>
        <li>Beginning HTML</li>
        <li>
                <ul>
                   <li>What HTML is</li>
                   <li>How to Write HTML</li>
                   <li>Doc structure</li>
                   <li>Headings</li>
                   <li>Paragraphs</li>
                   <li>Comments</li>
                </ul>
        </li>
        <li>Links</li>
        <li>More HTML</li>
</ol>
```

Many browsers format nested ordered lists and nested unordered lists differently from their enclosing lists. They might, for example, use a symbol other than a bullet for a nested list, or number the inner list with letters (`a`, `b`, `c`) rather than numbers. Don't

assume that this will be the case, however, and refer back to "section 8, subsection b" in your text because you can't determine what the exact formatting will be in the final output. If you do need to be sure which symbols or numbering scheme will be used for a list, specify a style using CSS.

The following input and output example shows a nested list and how it appears in a browser (see Figure 5.8):

Input ▼

```
<h1>Peppers</h1>
<ul>
  <li>Bell</li>
  <li>Chile</li>
  <li>
    <ul>
      <li>Serrano</li>
      <li>Jalapeno</li>
      <li>Habanero</li>
      <li>Anaheim</li>
    </ul>
  </li>
  <li>Szechuan</li>
  <li>Cayenne</li>
```

Output ▶

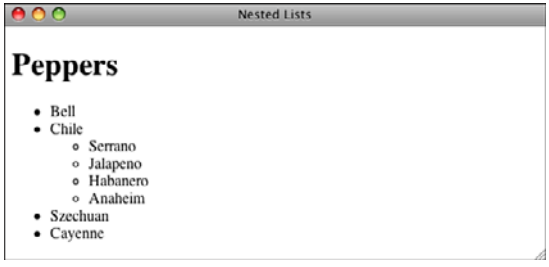


FIGURE 5.8
Nested lists.

DO	DON'T
Do remember that you can change the numbering and bullet styles for lists to suit your preference.	**Don't** use the deprecated list types; use one of the other lists instead.
Do feel free to nest lists to any extent that you like.	**Don't** number or format lists yourself; use the list tags.
	Don't use list tags to indent text on a page; use Cascading Style Sheets.

Other Uses for Lists

Lists have moved a long way past simple bullets. As it turns out, lists are useful when designing web pages because of the structure they provide. Semantically speaking, there are many common elements of web design that naturally lend themselves to list-like structures. Here are some advanced examples of how lists are used that combine a number of concepts that will be introduced throughout the book.

Many websites have lots of navigation links to present, and to keep from cluttering up the page, they use nested pull-down menus similar to those used in desktop applications. In this lesson, you've already seen that you can create nested lists in HTML. You can put your navigation links in such lists and then use CSS to radically change their appearance so that rather than looking like other lists, they instead look and behave like menus. There's an example of such menus in Figure 5.9.

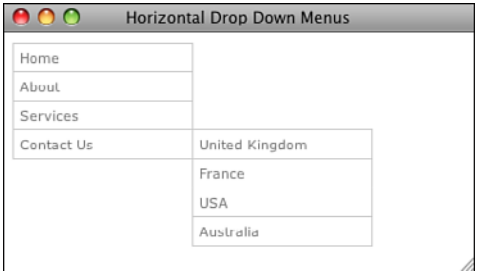


FIGURE 5.9
Pull-down navigation menus implemented using lists.

Using a combination of JavaScript and CSS, you can turn a standard HTML list into a sortable user interface element for a web application. You can see an example in Figure 5.10.

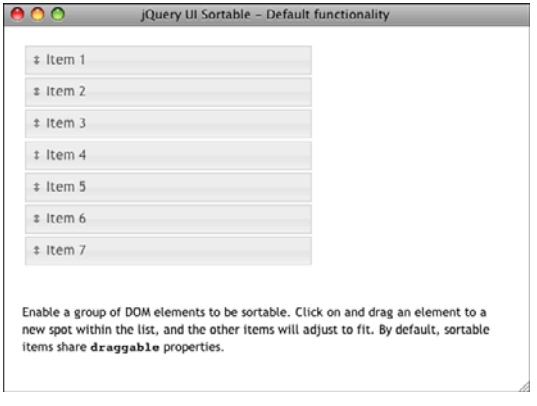


FIGURE 5.10
A sortable list.

You'll see other uses of lists in later lessons. With the introduction of CSS, lists became one of the fundamental building blocks of web pages.

Summary

In this brief lesson, you got a look at HTML lists. Lists are a core structural element for presenting content on web pages and can be used for everything from the list of steps in a process to a table of contents to a structured navigation system for a website. They come in three varieties: ordered lists, which are numbered; unordered lists, which by default are presented bullets; definition lists, which are presented as a series of terms and the definitions associated with them.

Not only are there CSS properties specifically associated with lists, but lists can also be styled using properties that apply to any block level element, like lists and list items.

The full list of HTML tags discussed in this lesson is shown in Table 5.2, and the CSS properties are shown in Table 5.3.

TABLE 5.2 HTML Tags from Lesson 5

Tag	Attribute	Use
`<ol>...</ol>`		An ordered (numbered) list. Each of the items in the list begins with `<li>`.
	`type`	Specifies the numbering scheme to use in the list. This attribute is deprecated in HTML 4.01.
	`start`	Specifies at which number to start the list. This attribute is deprecated in HTML 4.01.
`<ul>...</ul>`		An unordered (bulleted or otherwise marked) list. Each of the items in the list begins with `<li>`.
	`type`	Specifies the bulleting scheme to use in the list. This attribute is deprecated in HTML 4.01.
`<li>...</li>`		Individual list items in ordered, unordered, menu, or directory lists. The closing tag is optional in HTML, but is required in XHTML 1.0.
	`type`	Resets the numbering or bulleting scheme from the current list element. Applies only to `<ul>` and `<ol>` lists. This attribute is deprecated in HTML 4.01.

TABLE 5.2 Continued

Tag	Attribute	Use
	`value`	Resets the numbering in the middle of an ordered (`<ol>`) list. This attribute is deprecated in HTML 4.01.
`<dl>...</dl>`		A glossary or definition list. Items in the list consist of pairs of elements: a term and its definition.
`<dt>...</dt>`		The term part of an item in a glossary list. Closing tag is optional in HTML but required in XHTML 1.0.
`<dd>...</dd>`		The definition part of an item in a glossary list. Closing tag is optional in HTML but required in XHTML 1.0.

TABLE 5.3 CSS Properties from Lesson 5

Property	Use/Values
`list-style-type`	Used to specify the bullet style or numbering style for the list. Valid values are `disc`, `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, and `none`.
`list-style-image`	The image to use in place of the bullets for a list. The value should be the URL of the image.
`list-style-position`	Defines the alignment of lines of text in list items after the first. Values are `inside` and `outside`.
`list-style`	Enables you to set multiple list properties at once: list style type, list style position, and the URL of the bullet style.

Workshop

You've learned how to create and customize lists in HTML. In this section, you see the answers to some common questions about lists, and some exercises that should help you remember the things you've learned.

Q&A

Q My glossaries came out formatted really strangely! The terms are indented farther in than the definitions!

A Did you mix up the `<dd>` and `<dt>` tags? The `<dt>` tag is always used first (the definition term), and the `<dd>` follows (the definition). I mix them up all the time. There are too many `d` tags in glossary lists.

Q Is it possible to change the amount that list items are indented or remove the indentation entirely?

A Yes, the properties used to control list indentation are `margin-left` and `padding-left`. Some browsers use one and some use the other, so you need to set both of them to change the indentation for your lists. If you set them both to zero, the text in the list will be aligned with the left side of the container, and the bullets or numbers will actually be outside the container.

Q I've seen HTML files that use `<li>` outside a list structure, alone on the page, like this:

```
<li>And then the duck said, "put it on my bill"</li>
```

A Most browsers at least accept this tag outside a list tag and format it either as a simple paragraph or as a nonindented bulleted item. According to the true HTML specification, however, using an `<li>` outside a list tag is illegal, so good HTML pages shouldn't do this. Enclosing list items within list tags is also required by the XHTML recommendation. Always put your list items inside lists where they belong.

Quiz

1. Ordered and unordered lists use the `<li>` tag for list items; what tags are used by definition lists?
2. Is it possible to nest an ordered list within an unordered list or vice versa?
3. Which attribute is used to set the starting number for an ordered list? What about to change the value of an element within a list?
4. What are the three types of bullets that can be specified for unordered lists using the `list-style-type` CSS property?

Quiz Answers

1. Definition lists use the `<dt>` and `<dd>` tags for list items.
2. Yes, you can nest ordered lists within unordered lists or vice versa. You can also nest lists of the same type, too.
3. With the `<ol>` tag, the start attribute is used to specify the starting value for the list. To change the numbering within a list, the `value` attribute is used. Subsequent items are numbered from the most recent value that's specified.
4. The bullet types supported by the list-style-type property are `disc`, `circle`, and `square`. The default is `disc`.

Exercises

1. Use nested lists to create an outline of the topics covered in this book so far.
2. Use nested lists and the `list-style-type` CSS property to create a traditional outline of the topics you plan to cover on your own website.

LESSON 6

Adding Links to Your Web Pages

After finishing the preceding lesson, you now have a couple of pages that have some headings, text, and lists in them. These pages are all well and good but rather boring. The real fun starts when you learn how to create hypertext links and link your pages to the Web. In this lesson, you learn just that. Specifically, you learn about the following:

- All about the HTML link tag (<a>) and its various parts
- How to link to other pages on your local disk by using relative and absolute pathnames
- How to link to other pages on the Web by using URLs
- How to use links and anchors to link to specific places inside pages
- All about URLs: the various parts of the URL and the kinds of URLs you can use

Creating Links

To create a link in HTML, you need two things:

- The name of the file (or the URL of the file) to which you want to link
- The text that will serve as the clickable link

Only the text included within the link tag is actually visible on your page. When your readers click on the link, the browser loads the URL associated with the link.

The Link Tag: `<a>`

To create a link in an HTML page, you use the HTML link tag `<a>...</a>`. The `<a>` tag often is called an *anchor* tag because it also can be used to create anchors for links. (You'll learn more about creating anchors later in this lesson.) The most common use of the link tag, however, is to create links to other pages.

Unlike the simple tags you learned about in the preceding lesson, the `<a>` tag has some extra features: The opening tag, `<a>`, includes both the name of the tag (`a`) and extra information about the link itself. The extra features are called *attributes* of the tag. (You first discovered attributes in Lesson 5, "Organizing Information with Lists.") So, rather than the opening `<a>` tag having just a name inside brackets, it looks something like the following:

```
<a name="Up" href="menu.html" title="The Twelve Caesars">
```

The extra attributes (in this example, `name`, `href`, and `title`) describe the link itself. The attribute you'll probably use most often is the `href` attribute, which is short for *hypertext reference*. You use the `href` attribute to specify the name or URL of the file to which this link points.

Like most HTML tags, the link tag also has a closing tag, `</a>`. All the text between the opening and closing tags will become the actual link on the screen and be highlighted, underlined, or colored blue or red when the web page is displays. That's the text you or your readers will click to follow the link to the URL in the `href` attribute.

Figure 6.1 shows the parts of a typical link using the `<a>` tag, including the `href`, the text of the link, and the closing tag.

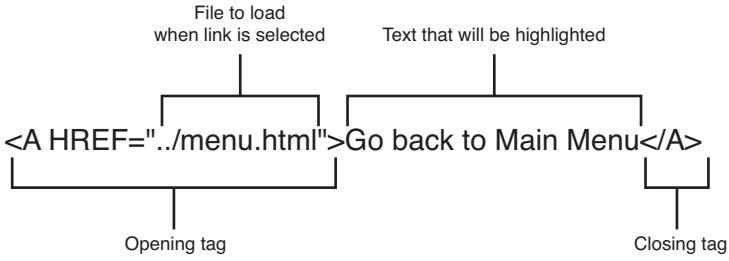


FIGURE 6.1
A link on a web page.

The following example shows a simple link and what it looks like (see Figure 6.2):

Input ▼

```
Go back to <a href="menu.html">Main Menu</a>
```

Output ▶

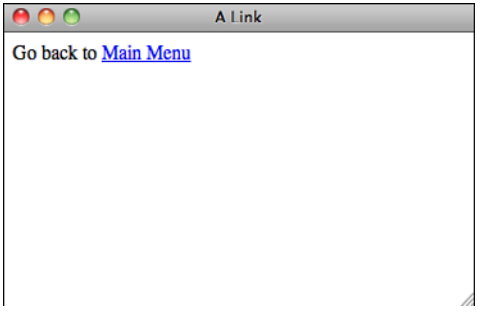


FIGURE 6.2
How a browser displays a link.

Task: Exercise 6.1: Linking Two Pages ▼

Now you can try a simple example with two HTML pages on your local disk. You need your text editor and your web browser for this exercise. Because both the pages you work with are on your local disk, you don't need to connect to the Internet. (Be patient; you'll get to do network stuff in the next section.)

Create two HTML pages and save them in separate files. Here's the code for the two HTML files for this section, called `menu.html` and `claudius.html`. What your two pages look like or what they're called doesn't matter. However, make sure that you insert your own filenames if you follow along with this example.

The following is the first file, called `menu.html`:

```
<!DOCTYPE html><html>
<head>
```

```
<title>The Twelve Caesars</title>
</head>
<body>
<h1>"The Twelve Caesars" by Suetonius</h1>
<p>Seutonius (or Gaius Suetonius Tranquillus) was born circa A.D. 70
and died sometime after A.D. 130. He composed a history of the twelve
Caesars from Julius to Domitian (died A.D. 96). His work was a
significant contribution to the best-selling novel and television
series "I, Claudius." Suetonius' work includes biographies of the
following Roman emperors:</p>
<ul>
 <li>Julius Caesar</li>
 <li>Augustus</li>
 <li>Tiberius</li>
 <li>Gaius (Caligula)</li>
 <li>Claudius</li>
 <li>Nero</li>
 <li>Galba</li>
 <li>Otho</li>
 <li>Vitellius</li>
 <li>Vespasian</li>
 <li>Titus</li>
 <li>Domitian</li>
</ul>
</body>
</html>
```

The list of menu items (Julius Caesar, Augustus, and so on) will be links to other pages. For now, just type them as regular text; you turn them into links later.

The following is the second file, `claudius.html`:

```
<!DOCTYPE html><html>
<head>
<title>The Twelve Caesars: Claudius</title>
</head>
<body>
<h2>Claudius Becomes Emperor</h2>
<p>Claudius became Emperor at the age of 50. Fearing the attack of
Caligula's assassins, Claudius hid behind some curtains. After a guardsman
discovered him, Claudius dropped to the floor, and then found himself
declared Emperor.</p>
<h2>Claudius is Poisoned</h2>
<p>Most people think that Claudius was poisoned. Some think his wife
Agrippina poisoned a dish of mushrooms (his favorite food). His death
was revealed after arrangements had been made for her son, Nero, to
succeed as Emperor.</p>
<p>Go back to Main Menu</p>
</body>
</html>
```

CAUTION

Make sure that both of your files are in the same directory or folder. If you haven't called them `menu.html` and `claudius.html`, make sure that you take note of the names because you'll need them later.

Create a link from the menu file to the feeding file. Edit the `menu.html` file, and put the cursor at the following line:

```
<li>Claudius</li>
```

Link tags don't define the format of the text itself, so leave in the list item tags and just add the link inside the item. First, put in the link tags (the `<a>` and `</a>` tags) around the text that you want to use as the link:

```
<li><a>Claudius</a></li>
```

Now add the name of the file that you want to link to as the `href` part of the opening link tag. Enclose the name of the file in quotation marks (straight quotes ["], not curly or typesetter's quotes [“]), with an equal sign between `href` and the name. Filenames in links are case-sensitive, so make sure that the filename in the link is identical to the name of the file you created. (`Claudius.html` is not the same file as `claudius.html`; it has to be exactly the same case.) Here I've used `claudius.html`; if you used different files, use those filenames.

```
<li><a href="claudius.html">Claudius</a></li>
```

Now, start your browser, select Open File (or its equivalent in your browser), and open the `menu.html` file. The paragraph you used as your link should now show up as a link that is in a different color, underlined, or otherwise highlighted. Figure 6.3 shows how it looked when I opened it.

Now, when you click the link, your browser should load and display the `claudius.html` page, as shown in Figure 6.4.

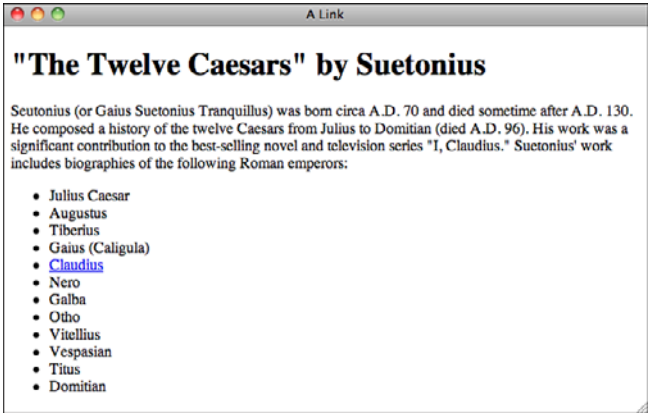


FIGURE 6.3
The *menu.html* file with link.

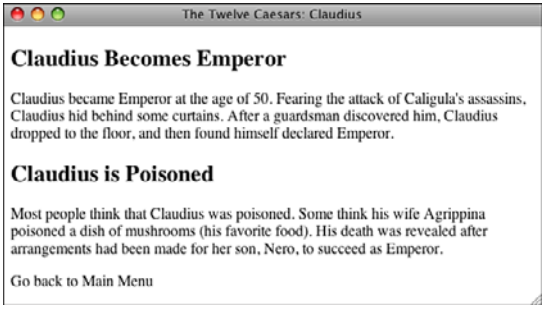


FIGURE 6.4
The *claudius.html* page.

If your browser can't find the file when you click on the link, make sure that the name of the file in the `href` part of the link tag is the same as the name of the file on the disk, uppercase and lowercase match, and both files are in the same directory. Remember to close your link, using the `</a>` tag, at the end of the text that serves as the link. Also, make sure that you have quotation marks at the end of the filename (sometimes you can easily forget) and that both quotation marks are ordinary straight quotes. All these things can confuse the browser and prevent it from finding the file or displaying the link properly.

Now you can create a link from the feeding page back to the menu page. A paragraph at the end of the `claudius.html` page is intended for just this purpose:

```
<p>Go back to Main Menu</p>
```

Add the link tag with the appropriate `href` to that line, such as the following in which `menu.html` is the original menu file:

```
<p><a href="menu.html">Go back to Main Menu</a></p>
```

Nesting Tags Properly

When you include tags inside other tags, make sure that the closing tag closes the tag that you most recently opened. That is, enter

```
<p> <a> .. </a> </p>
```

rather than

```
<p> <a> .. </p> </a>
```

Improper nesting of tags is invalid and could prevent your page from displayinf properly, so always make sure that you close the most recently opened tag first.

Now when you reload the Claudius file, the link will be active, and you can jump between the menu and the detail page by clicking on those links.

Linking Local Pages Using Relative and Absolute Pathnames

The example in the preceding section shows how to link together pages that are contained in the same folder or directory on your local disk (local pages). This section continues that thread, linking pages that are still on the local disk but might be contained in different directories or folders on that disk.

NOTE

Folders and directories are the same thing, but they're called different names depending on whether you're on Macintosh, Windows, or UNIX. I'll simply call them *directories* from now on to make your life easier.

When you specify just the filename of a linked file within quotation marks, as you did earlier, the browser looks for that file in the same directory as the current file. This is true even if both the current file and the file being linked to are on a server somewhere else on the Internet; both files are contained in the same directory on that server. It is the simplest form of a relative pathname.

Relative pathnames point to files based on their locations relative to the current file. They can include directory names, or they can point to the path you would take to navigate to that file if you started at the current directory or folder. A pathname might, for

example, include directions to go up two directory levels and then go down two other directories to get to the file.

To specify relative pathnames in links, you must use UNIX-style paths regardless of the system you actually have. You therefore separate directory or folder names with forward slashes (/), and you use two dots to refer generically to the directory above the current one (..).

Table 6.1 shows some examples of relative pathnames and where they lead.

TABLE 6.1 Relative Pathnames

Pathname	Means
`href="file.html"`	`file.html` is located in the current directory.
`href="files/file.html"`	`file.html` is located in the directory called `files`, and the `files` directory is located in the current directory.
`href="files/morefiles/file.html"`	`file.html` is located in the `morefiles` directory, which is located in the `files` directory, which is located in the current directory.
`href="../file.html"`	`file.html` is located in the directory one level up from the current directory (the parent directory).
`href="../../files/file.html"`	`file.html` is located two directory levels up, in the directory `files`.

If you're linking files on a personal computer (Macintosh or PC), and you want to link to a file on a different disk, use the name or letter of the disk as just another directory name in the relative path.

Absolute Pathnames

You can also specify the link to another page on your local system by using an absolute pathname.

Absolute pathnames point to files based on their absolute locations on the file system. Whereas relative pathnames point to the page to which you want to link by describing its location relative to the current page, absolute pathnames point to the page by starting at the top level of your directory hierarchy and working downward through all the intervening directories to reach the file.

Absolute pathnames always begin with a slash, which is the way they're differentiated from relative pathnames. Following the slash are all directories in the path from the top level to the file you are linking.

NOTE

Top has different meanings, depending on how you publish your HTML files. If you just link to files on your local disk, the top is the top of your file system (`/` on UNIX, or the disk name on a Macintosh or PC). When you publish files using a web server, the top is the directory where the files served by the web server are stored, commonly referred to as the *document root*. You learn more about absolute pathnames and web servers in Lesson 20, "Putting Your Site Online."

Table 6.2 shows some examples of absolute pathnames and what they mean.

TABLE 6.2 Absolute Pathnames Examples

Pathname	Means
`href="/u1/lemay/file.html"`	`file.html` is located in the directory `/u1/lemay` (typically on UNIX systems).
`href="/d¦/files/html/file.htm"`	`file.htm` is located on the `D:` disk in the directory `files/html` (on Windows systems).
`href="/Macintosh%20HD/HTML Files/file.html"`	`file.html` is located on the disk `Hard Disk 1`, in the folder `HTML Files` (typically on OS X systems).

Using Relative or Absolute Pathnames?

The answer to that question is, "It depends." If you have a set of files that link only to other files within that set, using relative pathnames makes sense. On the other hand, if the links in your files point to files that aren't within the same hierarchy, you probably want to use absolute links. Generally, a mix of the two types of links makes the most sense for complex sites.

Let's say that your site consists of two sections, `/stuff` and `/things`. If you want to link from the file `index.html` in `/stuff` to `history.html` in `/stuff` (or any other file in `/stuff`), you use a relative link. That way, you can move the `/stuff` directory around without breaking any of the internal links. On the other hand, if you want to create a link in `/stuff/index.html` to `/things/index.html`, an absolute link is probably called for. That way, if you move `/stuff` to `/more/stuff`, your link will still work.

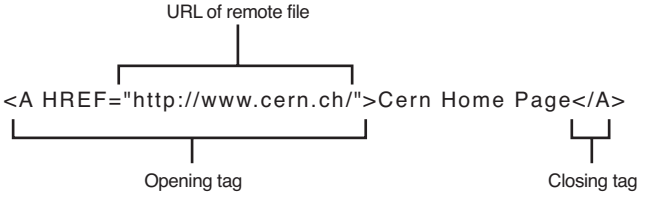
The rule of thumb I generally use is that if pages are part of the same collection, I use relative links, and if they're part of different collections, I use absolute links.

Links to Other Documents on the Web

So, now you have a whole set of pages on your local disk, all linked to each other. In some places in your pages, however, you want to refer to a page somewhere else on the Internet—for example, to The First Caesars page by Dr. Ellis Knox at Boise State University for more information on the early Roman emperors. You also can use the link tag to link those other pages on the Internet, which I'll call *remote* pages. *Remote pages* are contained somewhere on the Web other than the system on which you're currently working.

The HTML code you use to link pages on the Web looks exactly the same as the code you use for links between local pages. You still use the `<a>` tag with an `href` attribute, and you include some text to serve as the link on your web page. Rather than a filename or a path in the `href`, however, you use the URL of that page on the Web, as Figure 6.5 shows.

FIGURE 6.5
Link to remote files.



Task: Exercise 6.2: Linking Your Caesar Pages to the Web

Go back to those two pages you linked together earlier, the ones about the Caesars. The `menu.html` file contains several links to other local pages that provide information about 12 Roman emperors.

Now suppose that you want to add a link to the bottom of the menu file to point to The First Caesars page by Dr. Ellis Knox at Boise State University, whose URL is http://www.boisestate.edu/courses/westciv/julio-cl/.

First, add the appropriate text for the link to your menu page, as follows:

```
<p><i>The First Caesars</i> page by Dr. Ellis Knox has more information on
these Emperors.</p>
```

What if you don't know the URL of the home page for The First Caesars page (or the page to which you want to link), but you do know how to get to it by following several links on several different people's home pages? Not a problem. Use your browser to find the home page for the page to which you want to link. Figure 6.6 shows what The First Caesars page looks like in a browser.

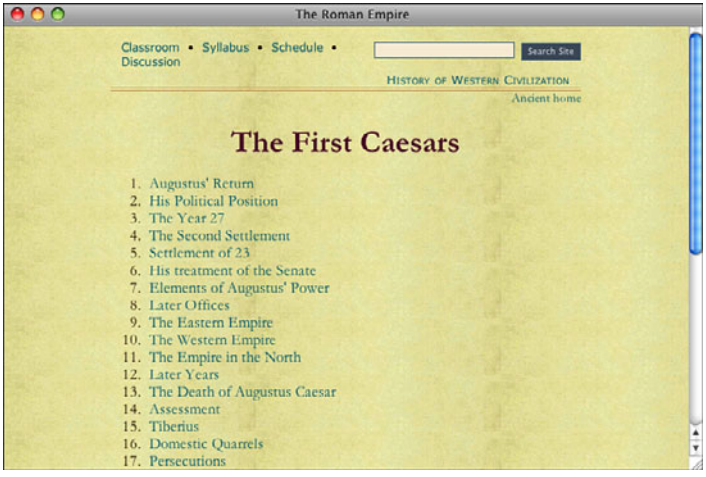


FIGURE 6.6
The First Caesars page.

NOTE

If your system isn't connected to the Internet, you might want to connect now so that you can test links to pages stored on the Web.

You can find the URL of the page you're currently viewing in your browser in the address box at the top of the browser window. To find the URL for a page you want to link to, use your browser to go to the page, copy the URL from the address field, and paste it into the `href` attribute of the link tag. No typing!

After you have the URL of the page, you can construct a link tag in your menu file and paste the appropriate URL into the link, like this:

Input

```
<p>"<i><a href="http://history.boisestate.edu/westciv/julio-cl/">
The First Caesars</a></i>" page by Dr. Ellis Knox has more information on these
Emperors.</p>
```

In that code I also italicized the title of the page using the `<i>` tag. You'll learn more about that tag and other text formatting tags on Lesson 7, "Formatting Text with HTML and CSS."

Of course, if you already know the URL of the page to which you want to link, you can just type it into the `href` part of the link. Keep in mind, however, that if you make a

mistake, your browser won't find the file on the other end. Many URLs are too complex for humans remember them; I prefer to copy and paste whenever I can to cut down on the chances of typing URLs incorrectly.

Figure 6.7 shows how the `menu.html` file, with the new link in it, looks when it displays.

Output ▶

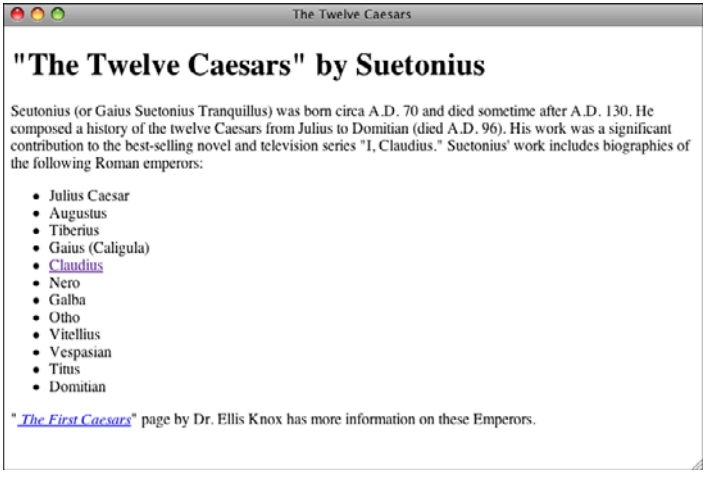


FIGURE 6.7
The First Caesars link.

▲

Task: Exercise 6.3: Creating a Link Menu

Now that you've learned how to create lists and links, you can create a *link menu*. Link menus are links on your web page that are arranged in list form or in some other short, easy-to-read, and easy-to-understand format. Link menus are terrific for pages that are organized in a hierarchy, for tables of contents, or for navigation among several pages. Web pages that consist of nothing but links often organize the links in menu form.

The idea of a link menu is that you use short, descriptive terms as the links, with either no text following the link or with a further description following the link itself. Link menus look best in a bulleted or unordered list format, but you also can use glossary lists or just plain paragraphs. Link menus enable your readers to scan the list of links quickly and easily, a task that might be difficult if you bury your links in body text.

In this exercise, you create a web page for a set of book reviews. This page serves as the index to the reviews, so the link menu you create is essentially a menu of book names.

Start with a simple page framework: a first-level heading and some basic explanatory text:

```
<!DOCTYPE html><html>
<head>
<title>Really Honest Book Reviews</title>
</head>
<body>
<h1>Really Honest Book Reviews</h1>
<p>I read a lot of books about many different subjects. Though I'm not a
book critic, and I don't do this for a living, I enjoy a really good read
every now and then. Here's a list of books that I've read recently:</p>
```

Now add the list that will become the links, without the link tags themselves. It's always easier to start with link text and then attach actual links afterward. For this list, use a tag to create a bulleted list of individual books. The `<ol>` tag wouldn't be appropriate because the numbers would imply that you were ranking the books in some way. Here's the HTML list of books, and Figure 6.8 shows the page as it currently looks with the introduction and the list:

Input ▼

```
<ul>
 <li><i>The Rainbow Returns</i> by E. Smith</li>
 <li><i>Seven Steps to Immeasurable Wealth</i> by R. U. Needy</li>
 <li><i>The Food-Lovers Guide to Weight Loss</i> by L. Goode</li>
 <li><i>The Silly Person's Guide to Seriousness</i> by M. Nott</li>
</ul>
</body>
</html>
```

Output ▶

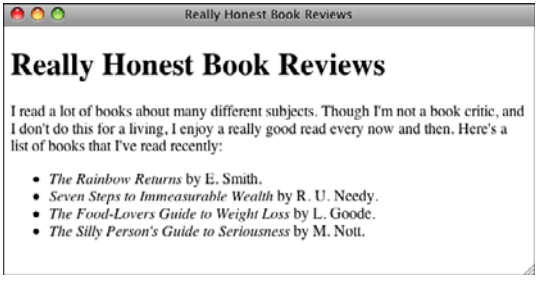


FIGURE 6.8
A list of books.

Now, modify each of the list items so that they include link tags. You need to keep the `<li>` tag in there because it indicates where the list items begin. Just add the `<a>` tags

▼ around the text itself. Here you link to filenames on the local disk in the same directory as this file, with each individual file containing the review for the particular book:

```
<ul>
 <li><a href="rainbow.html"><i>The Rainbow Returns</i> by E. Smith</a></li>
 <li><a href="wealth.html"><i>Seven Steps to Immeasurable Wealth</i> by R. U.
 Needy</a></li>
 <li><a href="food.html"><i>The Food-Lovers Guide to Weight Loss</i> by L.
 Goode</a></li>
 <li><a href="silly.html"><i>The Silly Person's Guide to Seriousness</i> by M.
 Nott</a></li>
</ul>
```

The menu of books looks fine, although it's a little sparse. Your readers don't know anything about each book (although some of the book names indicate the subject matter) or whether the review is good or bad. An improvement would be to add some short explanatory text after the links to provide hints of what is on the other side of the link:

Input ▼

```
<ul>
 <li><a href=rainbow.html"><i>The Rainbow Returns</i> by E. Smith</a>. A"
 fantasy story set in biblical times. Slow at times, but interesting.</li>
 <li><a href="wealth.html"><i>Seven Steps to Immeasurable Wealth</i> by R. U.
 Needy</a>. I'm still poor, but I'm happy! And that's the whole point.</li>
 <li><a href="food.html"><i>The Food-Lovers Guide to Weight Loss</i> by L. Goode
 </a>. At last! A diet book with recipes that taste good!</li>
 <li><a href="silly.html"><i>The Silly Person's Guide to Seriousness</i> by M.
 Nott</a>. Come on ... who wants to be serious?</li>
</ul>
```

The final list looks like Figure 6.9.

Output ▶

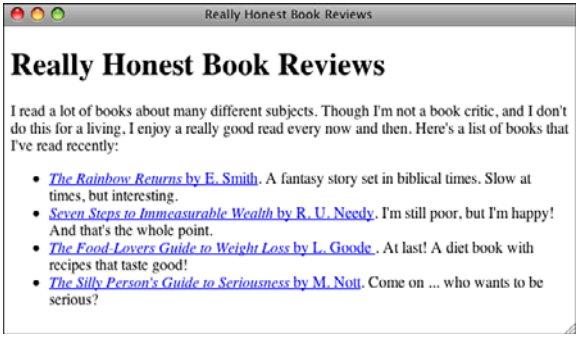


FIGURE 6.9
The final menu listing.

▲ You use link menus similar to this one throughout this book.

Linking to Specific Places Within Documents

The links you've created so far in this lesson have been from one point in a page to another page. But what if, rather than linking to that second page in general, you want to link to a specific place within that page—for example, to the fourth major section down?

You can do so in HTML by creating an anchor within the second page. The anchor creates a special element that you can link to inside the page. The link you create in the first page will contain both the name of the file to which you're linking and the name of that anchor. Then, when you follow the link with your browser, the browser will load the second page and then scroll down to the location of the anchor. (Figure 6.10 shows an example.)

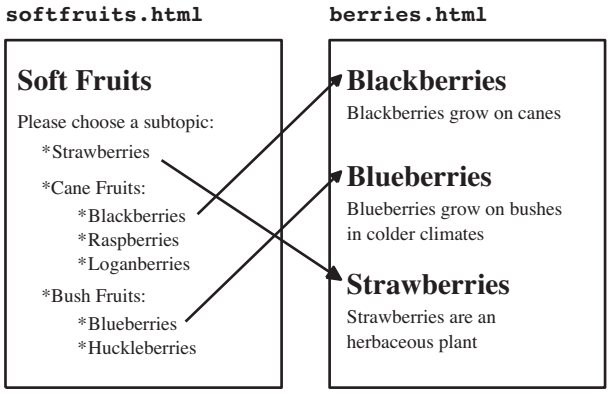


FIGURE 6.10
Links and anchors.

Anchors are special places that you can link to inside documents. Links can then jump to those special places inside the page as opposed to jumping just to the top of the page.

You can use links and anchors within the same page so that if you select one of those links, you jump to a different anchor within the page. For example, if you create an anchor at the top of a page, you could add links after each section of the page that return the user to the top. You could also create anchors at the beginning of each section and include a table of contents at the top of the page that has links to the sections.

Creating Links and Anchors

You create an anchor in nearly the same way that you create a link: by using the `<a>` tag. If you wondered why the link tag uses an `<a>` rather than an `<l>`, now you know: `a` actually stands for *anchor*.

When you specify links by using <a>, the link has two parts: the href attribute in the opening <a> tag, and the text between the opening and closing tags that serve as a hot spot for the link.

You create anchors in much the same way, but rather than using the href attribute in the <a> tag, you use the name attribute. The name attribute takes a keyword (or words) that name the anchor. Figure 6.11 shows the parts of the <a> tag when used to indicate an anchor.

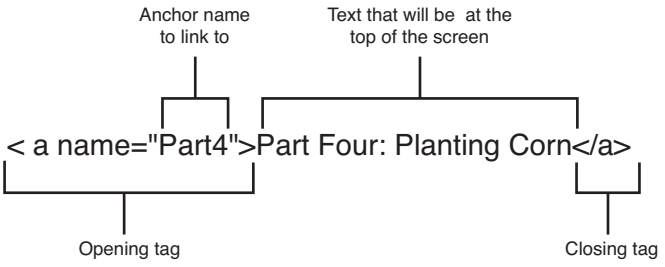


FIGURE 6.11
The <a> tag and anchors.

Including text between the anchor tags is optional. The actual anchor is placed at the location of the opening anchor tag, so you can just as easily write it as follows:

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

The browser scrolls the page to the location of the anchor so that it's at the top of the screen.

For example, to create an anchor at the section of a page labeled Part 4, you might add an anchor called part4 to the heading, similar to the following:

```
<h1><a name="part4">Part Four: Grapefruit from Heaven</a></h1>
```

Unlike links, anchors don't show up in the final displayed page. They're just a marker that links can point to.

To point to an anchor in a link, use the same form of link that you would when linking to the whole page, with the filename or URL of the page in the href attribute. After the name of the page, however, include a hash sign (#) and the name of the anchor exactly as it appears in the name attribute of that anchor (including the same uppercase and lowercase characters!), like the following:

```
<a href="mybigdoc.html#part4">Go to Part 4</a>
```

This link tells the browser to load the page mybigdoc.html and then to scroll down to the anchor named part4. The text inside the anchor definition will appear at the top of the screen.

Task: Exercise 6.4: Linking Sections Between Two Pages

Now let's create an example with two pages. These two pages are part of an online reference to classical music, in which each web page contains all the references for a particular letter of the alphabet (`a.html`, `b.html`, and so on). The reference could have been organized such that each section is its own page. Organizing it that way, however, would have involved several pages to manage, and many pages the readers would have to load if they were exploring the reference. Bunching the related sections together under lettered groupings is more efficient in this case. (Lesson 18, "Writing Good Web Pages: Do's and Don'ts," goes into more detail about the trade-offs between short and long pages.)

The first page you'll look at is for M; the first section looks like the following in HTML:

Input

```
<!DOCTYPE html><html>
<head>
<title>Classical Music: M</title>
</head>
<body>
<h1>M</h1>
<h2>Madrigals</h2>
<ul>
 <li>William Byrd, <em>This Sweet and Merry Month of May</em></li>
 <li>William Byrd, <em>Though Amaryllis Dance</em></li>
 <li>Orlando Gibbons, <em>The Silver Swan</em></li>
 <li>Claudio Monteverdi, <em>Lamento d'Arianna</em></li>
 <li>Thomas Morley, <em>My Bonny Lass She Smileth</em></li>
 <li>Thomas Weelkes, <em>Thule, the Period of Cosmography</em></li>
 <li>John Wilbye, <em>Sweet Honey-Sucking Bees</em></li>
</ul>
<p>Secular vocal music in four, five and six parts, usually a capella.
15th-16th centuries.</p>
<p><em>See Also</em>
Byrd, Gibbons, Lassus, Monteverdi, Morley, Weelkes, Wilbye </p>
</body
</html>
```

▼ Figure 6.12 shows how this section looks when it displays.

Output ▶

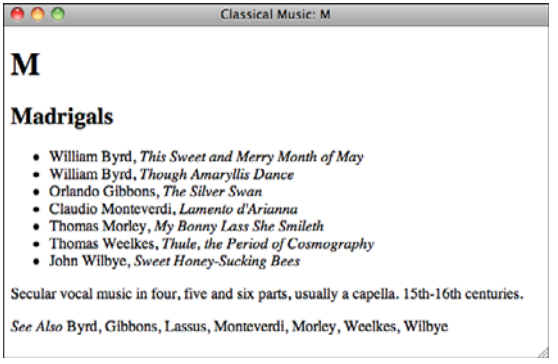


FIGURE 6.12
Part M of the Online Music Reference.

In the last line (the See Also), linking the composer names to their respective sections elsewhere in the reference would be useful. If you use the procedure you learned earlier, you can create a link here around the word `Byrd` to the page `b.html`. When your readers select the link to `b.html`, the browser drops them at the top of the Bs. Those hapless readers then have to scroll down through all the composers whose names start with *B* (and there are many of them: Bach, Beethoven, Brahms, Bruckner) to get to Byrd—a lot of work for a system that claims to link information so that you can find what you want quickly and easily.

What you want is to link the word `Byrd` in `m.html` directly to the section for Byrd in `b.html`. Here's the relevant part of `b.html` you want to link. (I've deleted all the Bs before `Byrd` to make the file shorter for this example. Pretend they're still there.)

NOTE

In this example, you see the use of the `<em>` tag. This tag is used to specify text that should be emphasized. The emphasis usually is done by rendering the text italic in the browser. I used it rather than the `<i>` tag because it describes the meaning I intend to convey rather than just describing a physical style for text on the page.

```
<!DOCTYPE html><html>
<head>
<title>Classical Music: B</title>
</head>
<body>
```

```
<h1>B</h1>
<!— I've deleted all the Bs before Byrd to make things shorter —>
<h2><a name="Byrd">Byrd, William, 1543-1623</a></h2>
<ul>
 <li>Madrigals
  <ul>
    <li><em>This Sweet and Merry Month of May</em></li>
    <li><em>Though Amaryllis Dance</em></li>
    <li><em>Lullabye, My Sweet Little Baby</em></li>
  </ul>
 </li>
 <li>Masses
  <ul>
    <li><em>Mass for Five Voices</em></li>
    <li><em>Mass for Four Voices</em></li>
    <li><em>Mass for Three Voices</em></li>
  </ul>
 </li>
 <li>Motets
  <ul>
    <li><em>Ave verum corpus a 4</em></li>
  </ul>
 </li>
</ul>
<p><em>See Also</em>
 Byrd, Gibbons, Lassus, Monteverdi, Morley, Weelkes, Wilbye</p>
</body>
</html>
```

You need to create an anchor at the section heading for Byrd. You then can link to that anchor from the See Also instances in the file for *M*.

As described earlier in this lesson, you need two elements for each anchor: an anchor name and the text inside the link to hold that anchor (which might be highlighted in some browsers). The latter is easy; the section heading itself works well because it's the element to which you're actually linking.

You can choose any name you want for the anchor, but each anchor in the page must be unique. (If you have two or more anchors with the name `fred` in the same page, how would the browser know which one to choose when a link to that anchor is selected?) A good, unique anchor name for this example is simply `byrd` because `byrd` can appear only one place in the file, and this is it.

After you decide on the two parts, you can create the anchor in your HTML file. Add the `<a>` tag to the `William Byrd` section heading, but be careful here. If you were working with normal text within a paragraph, you'd just surround the whole line with `<a>`. But when you're adding an anchor to a big section of text that's also contained within an

element—such as a heading or paragraph—always put the anchor inside the element. In other words, enter

```
<h2><a name="byrd">Byrd, William, 1543-1623</a></h2>
```

but do not enter

```
<a name="byrd"><h2>Byrd, William, 1543-1623</h2></a>
```

The second example can confuse your browser. Is it an anchor, formatted just like the text before it, with mysteriously placed heading tags? Or is it a heading that also happens to be an anchor? If you use the right code in your HTML file, with the anchor inside the heading, you avoid the confusion. The easiest answer is probably just putting the anchor ahead of the heading tag, like this:

```
<a name="byrd"></a>
<h2>Byrd, William, 1543-1623</h2>
```

So, you've added your anchor to the heading and its name is `"byrd"`. Now go back to the `See Also` line in your `m.html` file:

```
<p><em>See Also</em>
 Byrd, Gibbons, Lassus, Monteverdi, Morley, Weelkes, Wilbye</p>
```

You're going to create your link here around the word `Byrd`, just as you would for any other link. But what's the URL? As you learned previously, pathnames to anchors look similar to the following:

```
page_name#anchor_name
```

If you're creating a link to the `b.html` page, the `href` is as follows:

```
<a href="b.html">
```

Because you're linking to a section inside that page, add the anchor name to link that section so that it looks like this:

```
<a href="b.html#byrd">
```

Note the small `b` in `byrd`. Anchor names and links are case-sensitive; if you put `#Byrd` in your `href`, the link might not work properly. Make sure that the anchor name you use in the `name` attribute and the anchor name in the link after the `#` are identical.

CAUTION

A common mistake is to put a hash sign in both the anchor name and in the link to that anchor. You use the hash sign only to separate the page and the anchor in the link. Anchor names should never have hash signs in them.

So, with the new link to the new section, the `See Also` line looks like this:

```
<p><em>See Also</em>
 <a href="b.html#byrd">Byrd</a>,
 Gibbons, Lassus, Monteverdi, Morley, Weelkes, Wilbye</p>
```

Of course, you can go ahead and add anchors and links to the other parts of the reference for the remaining composers.

With all your links and anchors in place, test everything. Figure 6.13 shows the Madrigals section with the link to Byrd ready to be selected.

FIGURE 6.13
The Madrigals section with a link to Byrd.

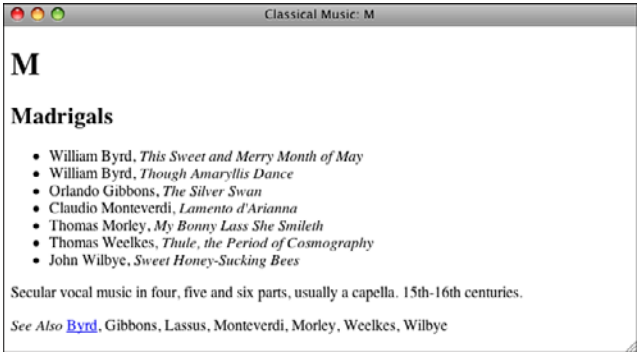


Figure 6.14 shows the screen that pops up when you select the Byrd link. If the page fits entirely within the window, the browser will not move down to the anchor, so you may need to reduce the size of your browser window to see how the link to the anchor takes you to the correct spot on the page.

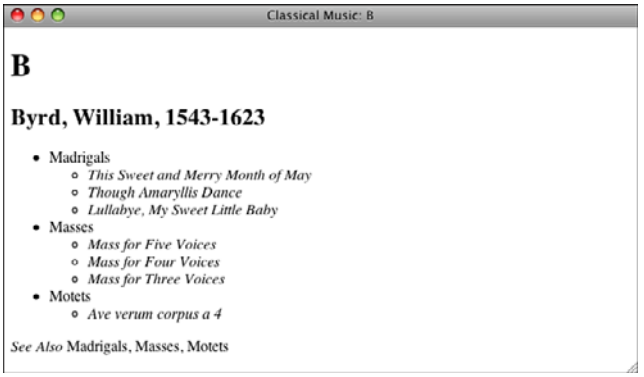


FIGURE 6.14
The Byrd section.

Linking to Anchors in the Same Document

What if you have only one large page, and you want to link to sections within that page? You can use anchors for it, too. For larger pages, using anchors can be an easy way to jump around within sections. To link to sections, you just need to set up your anchors at each section the way you usually do. Then, when you link to those anchors, leave off the name of the page itself but include the hash sign and the name of the anchor. So, if you're linking to an anchor name called `section5` in the same page as the link, the link looks like the following:

```
Go to <a href="#section5">The Fifth Section</a>
```

When you leave off the page name, the browser assumes that you're linking with the current page and scrolls to the appropriate section. You'll get a chance to see this feature in action in Lesson 7. There, you create a complete web page that includes a table of contents at the beginning. From this table of contents, the reader can jump to different sections in the same web page. The table of contents includes links to each section heading. In turn, other links at the end of each section enable the user to jump back to the table of contents or to the top of the page.

Anatomy of a URL

So far in this book, you've encountered URLs twice: in Lesson 1, "Navigating the World Wide Web," as part of the introduction to the Web; and in this lesson, when you created links to remote pages. If you've ever done much exploring on the Web, you've encountered URLs as a matter of course. You couldn't start exploring without a URL.

As I mentioned in Lesson 1, URLs are *uniform resource locators*. In effect, URLs are street addresses for bits of information on the Internet. Most of the time, you can avoid trying to figure out which URL to put in your links by just navigating to the bit of information you want with your browser, and then copying and pasting the long string of gobbledygook into your link. But understanding what a URL is all about and why it has to be so long and complex is often useful. Also, when you put your own information up on the Web, knowing something about URLs will be useful so that you can tell people where your web page is.

In this section, you learn what the parts of a URL are, how you can use them to get to information on the Web, and the kinds of URLs you can use (HTTP, FTP, Mailto, and so on).

Parts of URLs

Most URLs contain (roughly) three parts: the protocol, the hostname, and the directory or filename (see Figure 6.15).

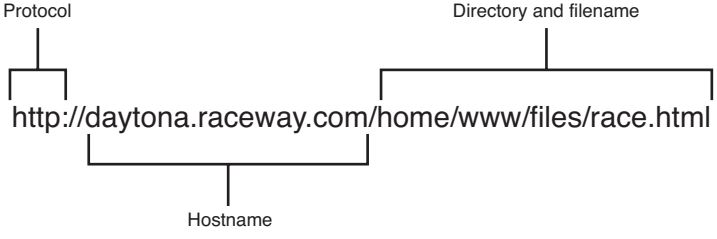


FIGURE 6.15
URL parts.

The *protocol* is the way in which the page is accessed; that is, the means of communication your browser uses to get the file. If the protocol in the URL is `http`, the browser will attempt to use the *Hypertext Transfer Protocol (HTTP)* to talk to the server. For a link to work, the host named in the link must be running a server that supports the protocol that's specified. So if you use an `ftp` URL to connect to www.example.com, the link won't work if that server isn't running *File Transfer Protocol (FTP)* server software.

The *hostname* is the address of the computer on which the information is stored, like www.google.com, ftp.apple.com, or www.aol.com. The same hostname can support more than one protocol, as follows:

```
http://exammple.com
ftp://example.com
```

It's one machine that offers two different information services, and the browser will use different methods of connecting to each. So long as both servers are installed and available on that system, you won't have a problem.

The hostname part of the URL might include a port number. The port number tells your browser to open a connection using the appropriate protocol on a specific network port. The only time you'll need a port number in a URL is if the server responding to the request has been explicitly installed on that port. If the server is listening on the default port, you can leave the port number out. This issue is covered in Lesson 19, "Designing for the Real World."

If a port number is necessary, it's placed after the hostname but before the directory, as follows:

```
http://my-public-access-unix.com:1550/pub/file
```

If the port is not included, the browser tries to connect to the default port number associated with the protocol in the URL. The default port for HTTP is 80, so a link to http://www.example.com:80/ and http://www.example.com/ are equivalent.

Finally, the *directory* is the location of the file or other form of information on the host. The directory does not necessarily point to a physical directory and file on the server. Some web applications generate content dynamically and just use the directory information as an identifier. For the files you work with while learning HTML, the directory information will point to files that exist on your computer.

Special Characters in URLs

A *special character* in a URL is anything that is not an upper- or lowercase letter, a number (0–9), or one of the following symbols: dollar sign (`$`), dash (`-`), underscore (_), or period (`.`). You might need to specify any other characters by using special URL escape codes to keep them from being interpreted as parts of the URL itself.

URL escape codes are indicated by a percent sign (`%`) and a two-character hexadecimal symbol from the ISO-Latin-1 character set (a superset of standard ASCII). For example, `%20` is a space, `%3f` is a question mark, and `%2f` is a slash. (Spaces are also sometimes encoded as + signs, and + signs are encoded as `%2b`.)

Suppose that you have a directory named `All My Files`. Your first pass at a URL with this name in it might look like the following:

```
http://myhost.com/harddrive/All My Files/www/file.html
```

If you put this URL in quotation marks in a link tag, it might work (but only if you put it in quotation marks). Because the spaces are considered special characters to the URL,

however, some browsers might have problems with them and not recognize the pathname correctly. For full compatibility with all browsers, use `%20`, as follows:

```
http://myhost.com/harddrive/A¦¦%20My%20Files/www/file.html
```

CAUTION

If you make sure that your file and directory names are short and use only alphanumeric characters, you won't need to include special characters in URLs. Special characters can be problematic in a variety of ways. When you're creating your own pages, you should avoid using spaces in file names as well as other non-alphanumeric characters whenever possible.The two exceptions are _ and -, which are the preferred separators between words in URLs.

Additional Attributes for the <a> Tag

There are some additional attributes for the <a> tag that are less common. These offer the following:

- **`tabindex`**—Supports a tabbing order so that authors can define an order for anchors and links, and then users can tab between them the way they do in a dialog box in Windows or the Mac OS.
- **Event handlers such as `onclick`, `onfocus`, and `onblur`**—The full list of events is included in the section " Global Attributes and Events " of Appendix B. You'll learn how to use these attributes on Lesson 14, "Introducing JavaScript."

Kinds of URLs

Many kinds of URLs are defined by the Uniform Resource Locator specification. (See Appendix A, "Sources of Further Information," for a pointer to the most recent version.) This section describes some of the more popular URLs and some situations to look out for when using them.

HTTP

HTTP URLs are by far the most common type of URLs because they point to other documents on the Web. HTTP is the protocol that World Wide Web servers use to communicate with web browsers.

HTTP URLs follow this basic URL form:

```
http://www.example.com/home/foo/
```

If the URL ends in a slash, the last part of the URL is considered a directory name. The file that you get using a URL of this type is the default file for that directory as defined by the HTTP server, usually a file called `index.html`. If the web page you're designing is the top-level file for all a directory's files, calling it `index.html` is a good idea. Putting such a file in place will also keep users from browsing the directory where the file is located.

You also can specify the filename directly in the URL. In this case, the file at the end of the URL is the one that is loaded, as in the following examples:

```
http://www.foo.com/home/foo/index.html
```

```
http://www.foo.com/home/foo/homepage.html
```

Using HTTP URLs such as the following, where `foo` is a directory, is also usually acceptable:

```
http://www.foo.com/home/foo
```

In this case, because `foo` is a directory, this URL should have a slash at the end. Most web servers can figure out that this is a link to a directory and redirect to the appropriate file.

Anonymous FTP

FTP URLs are used to point to files located on FTP servers—usually anonymous FTP servers; that is, the ones that allow you to log in using `anonymous` as the login ID and your email address as the password. FTP URLs also follow the standard URL form, as shown in the following examples:

```
ftp://ftp.foo.com/home/foo
ftp://ftp.foo.com/home/foo/homepage.html
```

Because you can retrieve either a file or a directory list with FTP, the restrictions on whether you need a trailing slash at the end of the URL aren't the same as with HTTP. The first URL here retrieves a listing of all the files in the `foo` directory. The second URL retrieves and parses the file `homepage.html` in the `foo` directory.

NOTE

Navigating FTP servers using a web browser can often be much slower than navigating them using FTP itself because the browser doesn't hold the connection open. Instead, it opens the connection, finds the file or directory listing, displays the listing, and then closes down the FTP connection. If you select a link to open a file or another directory in that listing, the browser constructs a new FTP URL from the items you selected, reopens the FTP connection by using the new URL, gets the next directory or file, and closes it again. For this reason, FTP URLs are best for when you know exactly which file you want to retrieve rather than for when you want to browse an archive.

Although your browser uses FTP to fetch the file, if it's an HTML file, your browser will display it just as it would if it were fetched using HTTP. Web browsers don't care how they get files. As long as they can recognize the file as HTML, either because the server explicitly says that the file is HTML or by the file's extension, browsers will parse and display that file as an HTML file. If they don't recognize it as an HTML file, no big deal. Browsers can either display the file if they know what kind of file it is or just save the file to disk.

Non-Anonymous FTP

All the FTP URLs in the preceding section are used for anonymous FTP servers. You also can specify an FTP URL for named accounts on an FTP server, like the following:

```
ftp://username:password@ftp.foo.com/home/foo/homepage.html
```

In this form of the URL, the `username` part is your login ID on the server, and `password` is that account's password. Note that no attempt is made to hide the password in the URL. Be very careful that no one is watching you when you're using URLs of this form—and don't put them into links that someone else can find!

Furthermore, the URLs that you request might be cached or logged somewhere, either on your local machine or on a proxy server between you and the site you're connecting to. For that reason, it's probably wise to avoid using this type of non-anonymous FTP URL altogether.

Mailto

The `mailto` URL is used to send electronic mail. If the browser supports `mailto` URLs, when a link that contains one is selected, the browser will prompt you for a subject and the body of the mail message, and send that message to the appropriate address when

you're done. Depending on how the user's browser and email client are configured, `mailto` links might not work at all for them.

The `mailto` URL is different from the standard URL form. It looks like the following:

```
mailto:internet_e-mail_address
```

Here's an example:

```
mailto:lemay@lne.com
```

NOTE

> If your email address includes a percent sign (`%`), you'll have to use the escape character `%25` instead. Percent signs are special characters to URLs.

Unlike the other URLs described here, the `mailto` URL works strictly on the client side. The `mailto` link just tells the browser to compose an email message to the specified address. It's up to the browser to figure out how that should happen. Most browsers will also let you add a default subject to the email by including it in the URL like this:

```
mailto:lemay@lne.com?subject=Hi there!
```

When the user clicks the link, most browsers will automatically stick `Hi there!` in the subject of the message. Some even support putting body text for the email message in the link, like this:

```
mailto:lemay@lne.com?subject=Hi there!&body=Body text.
```

File

File URLs are intended to reference files contained on the local disk. In other words, they refer to files located on the same system as the browser. For local files, URLs have an empty hostname (three slashes rather than two):

```
file:///dir1/dir2/file
```

You'll use `file` URLs a lot when you're testing pages you've created locally, although it's easier to use the browser's "Open File" functionality or drag and drop to open local files in your browser than it is to type in a `file` URL. Another use of file URLs is to create a local startup page for your browser with links to sites you use frequently. In this instance, because you'll be referring to a local file, using a file URL makes sense.

The problem with `file` URLs is that they reference local files, where *local* means on the same system as the browser pointing to the file—not the same system from which the

page was retrieved! If you use `file` URLs as links in your page, and someone from elsewhere on the Internet encounters your page and tries to follow those links, that person's browser will attempt to find the file on her local disk (and generally will fail). Also, because `file` URLs use the absolute pathname to the file, if you use file URLs in your page, you can't move that page elsewhere on the system or to any other system.

If your intention is to refer to files that are on the same file system or directory as the current page, use relative pathnames rather than file URLs. With relative pathnames for local files and other URLs for remote files, you shouldn't need to use a file URL at all.

Summary

In this lesson, you learned all about links. Links turn the Web from a collection of unrelated pages into an enormous, interrelated information system. (There are those big words again.)

To create links, you use the `<a>...</a>` tag pair, called the *link* or *anchor* tag. The anchor tag has attributes for creating links (the `href` attribute) and anchor names (the `name` attribute).

When linking pages that are all stored on the local disk, you can specify their pathnames in the `href` attribute as relative or absolute paths. For local links, relative pathnames are preferred because they enable you to move local pages more easily to another directory or to another system. If you use absolute pathnames, your links will break if you change anything in the hard-coded path.

If you want to link to a page on the Web (a remote page), the value of the `href` attribute is the URL of that page. You can easily copy the URL of the page you want to link. Just go to that page by using your favorite web browser, and then copy and paste the URL from your browser into the appropriate place in your link tag.

To create links to specific parts of a page, set an anchor at the point you want to link to, use the `<a>...</a>` tag as you would with a link, but rather than the `href` attribute, you use the `name` attribute to name the anchor. You then can link directly to that anchor name by using the name of the page, a hash sign (`#`), and the anchor name.

Finally, URLs (*uniform resource locators*) are used to point to pages, files, and other information on the Internet. Depending on the type of information, URLs can contain several parts, but most contain a protocol type and location or address. URLs can be used to point to many kinds of information but are most commonly used to point to web pages (`http`), FTP directories or files (`ftp`), or electronic mail addresses (`mailto`).

Workshop

Congratulations, you learned a lot in this lesson! Now it's time for the workshop. Many questions about links appear here. The quiz focuses on other items that are important for you to remember, followed by the quiz answers. In the following exercises, you take the list of items you created in Lesson 5, "Organizing Information with Lists," and link them to other pages.

Q&A

Q My links aren't being highlighted in blue or purple at all. They're still just plain text.

A Is the filename in a `name` attribute rather than in an `href`? Did you remember to close the quotation marks around the filename to which you're linking? Both of these errors can prevent links from showing up as links.

Q I put a URL into a link, and it shows up as highlighted in my browser, but when I click it, the browser says "unable to access page." If it can't find the page, why did it highlight the text?

A The browser highlights text within a link tag whether or not the link is valid. In fact, you don't even need to be online for links to show up as highlighted links, although you can't get to them. The only way you can tell whether a link is valid is to select it and try to view the page to which the link points.

As to why the browser couldn't find the page you linked to—make sure that you're connected to the network and that you entered the URL into the link correctly. Also verify that you have both opening and closing quotation marks around the filename, and that those quotation marks are straight quotes. If your browser prints link destinations in the status bar when you move the mouse cursor over a link, watch that status bar and see whether the URL that appears is actually the URL you want.

Finally, try opening the URL directly in your browser and see whether that solution works. If directly opening the link doesn't work either, there might be several reasons why. The following are two common possibilities:

- The server is overloaded or is not on the Internet.

  Machines go down, as do network connections. If a particular URL doesn't work for you, perhaps something is wrong with the machine or the network. Or maybe the site is popular, and too many people are trying to access it simultaneously. Try again later. If you know the people who run the server, you can try sending them electronic mail or calling them.

- The URL itself is bad.

  Sometimes URLs become invalid. Because a URL is a form of absolute pathname, if the file to which it refers moves around, or if a machine or directory name gets changed, the URL won't be valid anymore. Try contacting the person or site you got the URL from in the first place. See whether that person has a more recent link.

  Be sure to read the error message provided by the browser carefully. Often it will describe the reason why the link can't be opened, indicating whether it is a network problem or a problem with the URL.

Q Can I put any URL in a link?

A You bet. If you can get to a URL using your browser, you can put that URL in a link. Note, however, that some browsers support URLs that others don't. For example, Lynx is good with mailto URLs (URLs that enable you to send electronic mail to a person's email address). When you select a mailto URL in Lynx, it prompts you for a subject and the body of the message. When you're done, it sends the mail.

Q Can I use images as links?

A Yup, in more ways than one, actually. You learn how to use images as links and define multiple links within one image using image maps in Lesson 9, "Adding Images, Color, and Backgrounds."

Q My links aren't pointing to my anchors. When I follow a link, I'm always dropped at the top of the page rather than at the anchor. What's going on here?

A Are you specifying the anchor name in the link after the hash sign the same way that it appears in the anchor itself, with all the uppercase and lowercase letters identical? Anchors are case-sensitive, so if your browser can't find an anchor name with an exact match, the browser might try to select something else in the page that's closer. This is dependent on browser behavior, of course, but if your links and anchors aren't working, the problem usually is that your anchor names and your anchors don't match. Also, remember that anchor names don't contain hash signs—only the links to them do.

Q Is there any way to indicate a subject in a mailto URL?

A If you include `?subject=Your%20subject` in the mailto URL, it will work with most email clients. Here's what the whole link looks like:

```
<a href="mailto:someone@example.com?subject=Your%20subject">Send email</a>
```

Quiz

1. What two things do you need to create a link in HTML?
2. What's a relative pathname? Why is it advantageous to use them?
3. What's an absolute pathname?
4. What's an anchor, and what is it used for?
5. Besides HTTP (web page) URLs, what other kinds are there?

Quiz Answers

1. To create a link in HTML, you need the name or URL of the file or page to which you want to link, and the text that your readers can select to follow the link.
2. A relative pathname points to a file, based on the location that's relative to the current file. Relative pathnames are portable, meaning that if you move your files elsewhere on a disk or rename a directory, the links require little or no modification.
3. An absolute pathname points to a page by starting at the top level of a directory hierarchy and working downward through all intervening directories to reach the file.
4. An anchor marks a place that you can link to inside a web document. A link on the same page or on another page can then jump to that specific location instead of the top of the page.
5. Other types of URLs are FTP URLs (which point to files on FTP servers); `file` URLs (which point to a file contained on a local disk); and `mailto` URLs (which are used to send electronic mail).

Exercises

1. Remember the list of topics that you created in Lesson 5 in the first exercise? Create a link to the page you created in Lesson 5's second exercise (the page that described one of the topics in more detail).
2. Now, open the page that you created in Lesson 5's second exercise, and create a link back to the first page. Also, find some pages on the World Wide Web that discuss the same topic and create links to those pages, too. Good luck!

LESSON 7

Formatting Text with HTML and CSS

Over the previous three lessons, you learned the basics of HTML, including tags used to create page structure and add links. With that background, you're now ready to learn more about what HTML and CSS can do in terms of text formatting and layout. In this lesson, you learn about most of the remaining tags in HTML that you need to know to construct pages, including how to use HTML and CSS to do the following:

- Specify the appearance of individual characters (bold, italic, underlined)
- Include special characters (characters with accents, copyright marks, and so on)
- Create preformatted text (text with spaces and tabs retained)
- Align text left, right, and centered
- Change the font and font size
- Create other miscellaneous HTML text elements, including line breaks, rule lines, addresses, and quotations

Character-Level Elements

When you use HTML tags to create paragraphs, headings, or lists, those tags affect that block of text as a whole—changing the font, changing the spacing above and below the line, or adding characters (in the case of bulleted lists). They're referred to as *block-level elements*.

Character-level elements are tags that affect words or characters within other HTML tags and change the appearance of that text so that it's somehow different from the surrounding text—making it bold or underlined, for example.

To change the appearance of a set of characters within text, you can use one of two kinds of tags: logical styles or physical styles.

Logical Styles

Logical style tags describe the meaning of the text within the tag, not how it should look in the browser. They're similar to the common element tags for paragraphs or headings. For example, logical style tags might indicate a definition, a snippet of code, or an emphasized word. This can be a bit confusing because there are de facto standards that correlate each of these tags with a certain visual style. In other words, even though a tag like `<strong>` would mean different things to different people, in most browsers it means boldface.

Using logical style tags, the browser determines the actual presentation of the text, whether it's bold, italic, or any other change in appearance. You cannot guarantee that text that's highlighted using these tags will always be bold or italic, so you shouldn't depend on it. These days, browser makers have pretty much agreed on how each of these logical tags are rendered, but it's still important to understand that the logical tags convey more meaning than just the physical styles that they apply.

Each character style tag has both opening and closing sides and affects the text within those two tags. The following are the eight logical style tags:

`<em>` This tag indicates that the characters are to be emphasized in some way; that is, they're formatted differently from the rest of the text. In graphical browsers, typically `<em>` italicizes the text. For example:

```
<p>The anteater is the <em>strangest</em> looking animal,
isn't it?</p>
```

`<strong>` With this tag, the characters are to be more strongly emphasized than with `<em>`—usually in boldface. Consider the following:

```
<p>Take a <strong>left turn</strong> at <strong>Dee's Hop
Stop</strong></p>
```

`<code>` This tag indicates that the text inside is a code sample and displays it in a fixed-width font such as Courier. For example:

```
<p><code>#include "trans.h"</code></p>
```

`<samp>` This tag indicates sample text and is generally presented in a fixed-width font, like `<code>`. An example of its usage follows:

```
<p>The URL for that page is <samp>http://www.cern.ch/
</samp></p>
```

`<kbd>` This tag indicates text that's intended to be typed by a user. It's also presented in a fixed-width font. Consider the following:

```
<p>Type the following command: <kbd>find . -name "prune"
-print</kbd></p>
```

`<var>` This tag indicates the name of a variable, or some entity to be replaced with an actual value. Often it's displayed as italic or underline, and is used as follows:

```
<p><code>chown</code> <var>your_name for the_file
</var></p>
```

`<dfn>` This tag indicates a definition. `<dfn>` is used to highlight a word (usually in italics) that will be defined or has just been defined, as in the following example:

```
<p>Styles that are named after how they are actually
used are called
<dfn>logical styles</dfn></p>
```

`<cite>` This tag indicates a short quote or citation, as in the following:

```
<p>Eggplant has been known to cause nausea in some
people<cite> (Lemay, 1994)</cite></p>
```

HTML 4.01 introduced two additional logical style tags that are most useful for audio browsers. A graphical browser, such as Firefox or Internet Explorer, won't display them any differently. When an audio browser reads content included within one of these tags, however, each letter is spoken individually. For example, *fox* is pronounced F-O-X rather than *fox*.

These tags also use opening and closing sides and affect the text within. The following are new tags:

`<abbr>` This tag indicates the abbreviation of a word, as in the following:

```
<p>Use the standard two-letter state abbreviation
(such as <abbr>CA</abbr> for California)</p>
```

`<acronym>`	Similar to the `<abbr>` tag, `<acronym>` designates a word formed by combining the initial letters of several words, as in the following example:

```
<p>Jonathan learned his great problem-handling skills
from <acronym>STEPS</acronym> (Simply Tackle Each Problem
Seriously)</p>
```

Only the `<abbr>` tag made it into HTML5; `<acronym>` has been removed due to redundancy. You may still see it used but you should use the `<abbr>` tag instead. The following code snippets demonstrate each of the logical style tags, and Figure 7.1 illustrates how all the tags display.

Input ▼

```
<p>The anteater is the <em>strangest</em> looking animal, isn't it?</p>
<p>Take a <strong>left turn</strong> at <strong>Dee's Hop Stop
</strong></p>
<p><code>#include "trans.h"</code></p>
<p>The URL for that page is <samp>http://www.cern.ch/</samp></p>
<p>Type the following command: <kbd>find . -name "prune" -print</kbd></p>
<p><code>chown </code><var>your_name the_file</var></p>
<p>Styles that are named after how they are used are called <dfn>logical
styles</dfn></p>
<p>Eggplant has been known to cause nausea in some
people<cite> (Lemay, 1994)</cite></p>
<p>Use the standard two-letter state abbreviation (such as
<abbr>CA</abbr> for California)</p>
<p>Jonathan learned his great problem-handling skills from
<acronym>STEPS</acronym> (Simply Tackle Each Problem Seriously)</p>
```

Output ▶

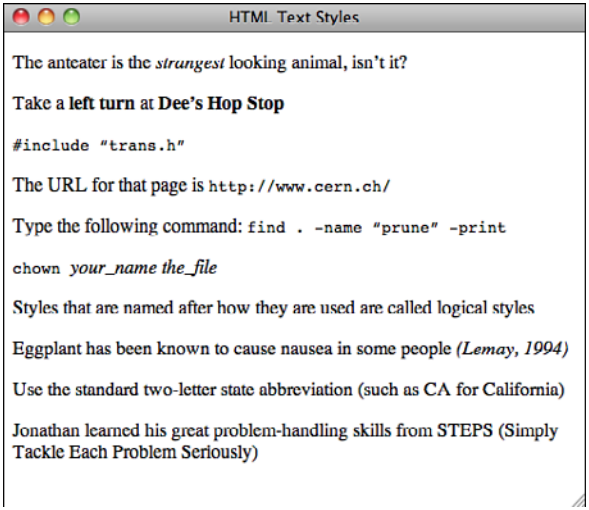


FIGURE 7.1
Various logical styles displayed in a browser.

Physical Styles

Over time, a number of physical style tags were added to HTML as well, but most of them have been removed from HTML5. You may see them used but avoid using them and use CSS instead.

`<b>`	Bold
`<i>`	Italic
`<tt>`	Monospaced typewriter font (removed from HTML5)
`<u>`	Underline (removed from HTML5)
`<s>`	Strikethrough (removed from HTML5)
`<big>`	Bigger print than the surrounding text (removed from HTML5)
`<small>`	Smaller print
`<sub>`	Subscript
`<sup>`	Superscript

NOTE

Text-based browsers, such as Lynx and those associated with some mobile devices, can't render bold, italic, or other styled text. They generally highlight the text in some way, but the method varies depending on the browser and platform.

You can nest character tags—for example, using both bold and italic for a set of characters—as follows:

```
<b><i>Text that is both bold and italic</i></b>
```

However, the result on the screen is browser-dependent, like all HTML tags. You won't necessarily end up with text that's both bold and italic. You might end up with one style or the other:

Input ▼

```
<p>In Dante's <i>Inferno</i>, maladoge was the eighth circle of hell,
and held the malicious and fraudulent.</p>
<p>All entries must be received by <b>September 26, 1999</b>.</p>
<p>Type <tt>lpr -Pbirch myfile.txt</tt> to print that file.</p>
<p>Sign your name in the spot marked <u>Sign Here</u>:</p>
<p>People who wear orange shirts and plaid pants <s>have no taste</s>
are fashion-challenged.</p>
<p>RCP floor mats give you <big>big</big> savings over the
competition!</p>
<p>Then, from the corner of the room, he heard a <small>tiny voice
</small>.</p>
<p>In heavy trading today. Consolidated Orange Trucking
rose <sup>1</sup>/<sub>4</sub>
points on volume of 1,457,900 shares.</p>
```

Figure 7.2 shows some of the physical tags and how they appear.

Output ▶

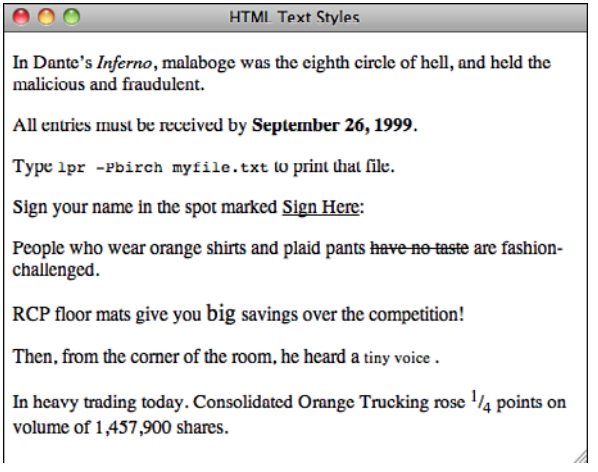


FIGURE 7.2
Physical styles displayed in a browser.

Character Formatting Using CSS

You've already seen how styles can modify the appearance of various elements. Any of the effects associated with the tags introduced in this lesson can also be created using CSS. Before I go into these properties, however, I want to talk a bit about how to use them. As I've said before, the `style` attribute can be used with most tags. However, most tags somehow affect the appearance of the text that they enclose. There's a tag that doesn't have any inherent effect on the text that it's wrapped around: the `<span>` tag. It exists solely to be associated with style sheets. It's used exactly like any of the other tags you've seen in this lesson. Simply wrap it around some text, like this:

```
<p>This is an example of the <span>usage of the span tag</span>.</p>
```

Used by itself, the `<span>` tag has absolutely no effect. Paired with the `style` attribute, it can take the place of any of the tags you've seen in this lesson and can do a lot more than that, as well.

The Text Decoration Property

The `text-decoration` property is used to specify which, if any, decoration will be applied to the text within the affected tag. The valid values for this property are `underline`, `overline`, `line-through`, and `blink`. The application of each of them is self-explanatory. However, here's an example that demonstrates how to use each of them:

```
<p>Here is some <span style='text-decoration: underline">underlined
text</span>.</p>
<p>Here is some <span style="text-decoration: overline">overlined
text</span>.</p>
<p>Here is some <span style="text-decoration: line-through">line-through
text</span>.</p>
<p>Here is some <span style="text-decoration: blink">blinking text</span>.</p>
```

The cool thing is that you can use this, and all the properties you'll see in this lesson, with any tag that contains text. Take a look at this example:

```
<h1 style="text-decoration: underline">An Underlined Heading</h1>
```

Using the `style` attribute, you can specify how the text of the heading appears. Choosing between this approach and the `<u>` tag is a wash—if you want to remove the underlining from the heading, you have to come back and edit the tag itself, regardless of whether you used the `<u>` tag or style attribute. Later, you'll see how to use style sheets to control the appearance of many elements simultaneously.

Font Properties

When you want to modify the appearance of text, the other major family of properties you can use is font properties. You can use font properties to modify pretty much any aspect of the type used to render text in a browser. One of the particularly nice things about font properties is that they're much more specific than the tags that you've seen so far.

First, let's look at some of the direct replacements for tags you've already seen. The `font-style` property can be used to italicize text. It has three possible values: `normal`, which is the default; `italic`, which renders the text in the same way as the `<i>` tag; and `oblique`, which in theory is somewhere between italic and normal, and is rendered by nearly all browsers as regular italics. Here are some examples:

```
<p>Here's some <span style="font-style: italic">italicized text</span>.</p>
<p>Here's some <span style="font-style: oblique">oblique text</span>
(which may look like regular italics in your browser).</p>
```

Now let's look at how you use CSS to create boldfaced text. In the world of HTML, you have two options: bold and not bold. With CSS, you have (theoretically) many more options. The reason I say *theoretically* is that browser support for the wide breadth of font weights available using CSS can be spotty. To specify that text should be boldface, the `font-weight` property is used. Valid values are `normal` (the default), `bold`, `bolder`, `lighter`, and 100 through 900, in units of 100. Here are some examples:

```
<p>Here's some <span style="font-weight: bold">bold text</span>.</p>
<p>Here's some <span style="font-weight: bolder">bolder text</span>.</p>
<p>Here's some <span style="font-weight: lighter">lighter text</span>.</p>
<p>Here's some <span style="font-weight: 700">bolder text</span>.</p>
```

You can also set the typeface for text using the `font-family` property. In addition, you can set the specific font for text, but I'm not going to discuss that until later in the lesson. In the meantime, let's look at how you can set the font to a member of a particular font family. The specific font will be taken from the user's preferences. The property to modify is `font-family`. The possible values are `serif`, `sans-serif`, `cursive`, `fantasy`, and `monospace`. So, if you want to specify that a monospace font should be used with CSS rather than the `<tt>` tag, you use the following code:

```
<p><span style="font-family: monospace">This is monospaced text.</span></p>
```

Now let's look at one capability not available using regular HTML tags. Using the `font-variant` property, you can have your text rendered so that lowercase letters are replaced with larger capital letters. The two values available are `normal` and `small-caps`. Here's an example:

```
<p><span style='font-variant: small-caps'>This Text Uses Small Caps.</span></p>
```

The web page in Figure 7.3 contains some text that uses the `font-variant` property as well as all the other properties described in this section.

Output ▶

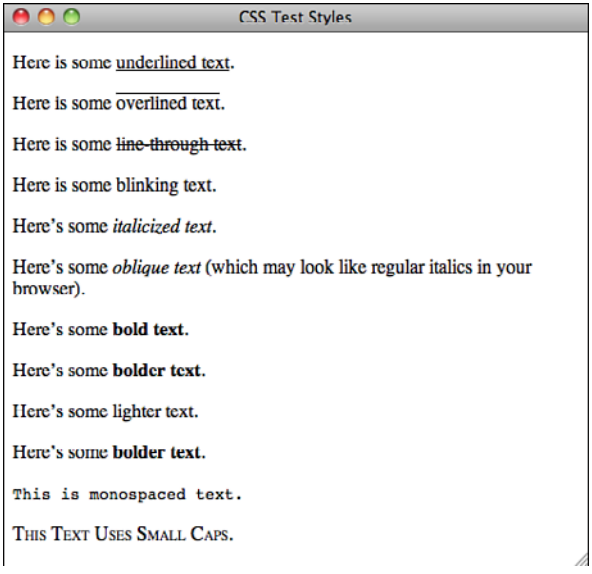


FIGURE 7.3
Text styled using CSS.

Preformatted Text

Most of the time, text in an HTML file is formatted based on the HTML tags used to mark up that text. In Lesson 3, "Introducing HTML and XHTML," I mentioned that any extra whitespace (spaces, tabs, returns) that you include in your HTML source is stripped out by the browser.

The one exception to this rule is the preformatted text tag `<pre>`. Any whitespace that you put into text surrounded by the `<pre>` and `</pre>` tags is retained in the final output. With these tags, the spacing in the text in the HTML source is preserved when it's displayed on the page.

The catch is that preformatted text usually is displayed (in graphical displays, at least) in a monospaced font such as Courier. Preformatted text is excellent for displaying code examples in which you want the text formatted with exactly the indentation the author used. Because you can use the `<pre>` tag to align text by padding it with spaces, you can use it for simple tables. However, the fact that the tables are presented in a monospaced font might make them less than ideal. (You'll learn how to create real tables in Lesson 10, "Building Tables.") The following is an example of a table created with `<pre>`:

Input ▼

```
<pre>
          Diameter    Distance      Time to    Time to
          (miles)     from Sun      Orbit      Rotate
                      (millions
                      of miles)
-----------------------------------------------------------
Mercury    3100         36           88 days    59 days
Venus      7700         67          225 days   244 days
Earth      7920         93          365 days    24 hrs
Mars       4200        141          687 days    24 hrs 24 mins
Jupiter   88640        483         11.9 years    9 hrs 50 mins
Saturn    74500        886         29.5 years   10 hrs 39 mins
Uranus    32000       1782           84 years   23 hrs
Neptune   31000       2793          165 days    15 hrs 48 mins
Pluto      1500       3670          248 years    6 days 7 hrs
</pre>
```

Figure 7.4 shows how it looks in a browser.

Output ▶

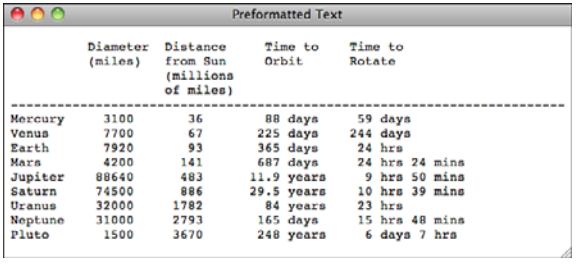


FIGURE 7.4
A table created using *<pre>*, shown in a browser.

When you're creating text for the <pre> tag, you can use link tags and character styles but not element tags such as headings or paragraphs. You should break your lines with hard returns and try to keep your lines to 60 characters or fewer. Some browsers might have limited horizontal space in which to display text. Because browsers usually won't reformat preformatted text to fit that space, you should make sure that you keep your text within the boundaries to prevent your readers from having to scroll from side to side.

Be careful with tabs in preformatted text. The actual number of characters for each tab stop varies from browser to browser. One browser might have tab stops at every fourth character, whereas another may have them at every eighth character. You should convert any tabs in your preformatted text to spaces so that your formatting isn't messed up if it's viewed with different tab settings than in the program you used to enter the text.

The <pre> tag is also excellent for converting files that were originally in some sort of text-only form, such as mail messages or Usenet news postings, into HTML quickly and easily. Just surround the entire content of the article within <pre> tags and you have instant HTML, as in the following example:

```
<pre>
To: lemay@lne.com
From: jokes@lne.com
Subject: Tales of the Move From Hell, pt. 1

I spent the day on the phone today with the entire household
services division of northern California, turning off services,
turning on services, transferring services and other such fun
things you have to do when you move.

It used to be you just called these people and got put on hold for
and interminable amount of time, maybe with some nice music, and
then you got a customer representative who was surly and hard of
hearing, but with some work you could actually get your phone
turned off.
</pre>
```

One creative use of the <pre> tag is to create ASCII art for your web pages. The following HTML input and output example shows a simple ASCII-art cow:

Input ▼

```
<pre>
   ( )
Moo (oo)
   \/———\
    ||    | \
    ||—-W|| *
    ||    ||
    ||    ||
</pre>
```

Figure 7.5 displays the result.

Output ▶

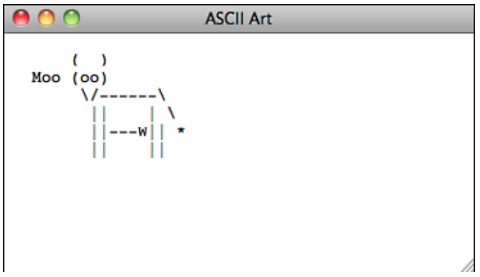


FIGURE 7.5
A bit of ASCII art that illustrates how preformatted text works.

Horizontal Rules

The <hr> tag, which has no closing tag in HTML and no text associated with it, creates a horizontal line on the page. Rule lines are used to visually separate sections of a web page—just before headings, for example, or to separate body text from a list of items.

Closing Empty Elements

The <hr> tag has no closing tag in HTML. To convert this tag to XHTML and to ensure compatibility with HTML browsers, add a space and a forward slash to the end of the tag:

```
<hr />
```

If the horizontal line has attributes associated with it, the forward slash still appears at the end of the tag, as shown in the following examples:

```
<hr size="2" />
<hr width="75%" />
<hr align="center" size="4" width="200" />
```

The following input shows a rule line and a list as you would write it in XHTML 1.0:

Input ▼

```
<hr />
<h2>To Do on Friday</h2>
<ul>
<li>Do laundry</li>
<li>Send FedEx with pictures</li>
<li>Have lunch with Mollie</li>
<li>Read Email</li>
<li>Set up Ethernet</li>
</ul>
<hr />
```

Figure 7.6 shows how they appear in a browser.

Output ▶

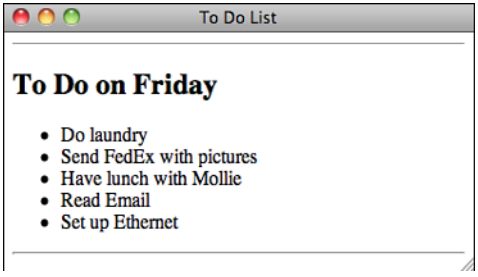


FIGURE 7.6
An example of how horizontal rules are used around a list.

Attributes of the `<hr>` Tag

There are a number of attributes that can be used to modify the appearance of a horizontal rule. Although they work in current browsers, they have been removed from HTML5, and you should use CSS instead. The `size` attribute indicates the thickness, in pixels, of the rule line. The default is `2`, and this also is the smallest that you can make the rule line. Figure 7.7 shows the sample rule line thicknesses created with the following code:

Input ▼

```
<h2>2 Pixels</h2>
<hr size="2" />
<h2>4 Pixels</h2>
<hr size="4" />
<h2>8 Pixels</h2>
<hr size="8" />
<h2>16 Pixels</h2>
<hr size="16" />
```

Output ▶

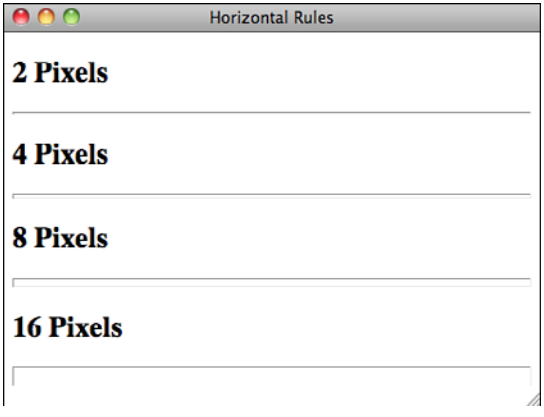


FIGURE 7.7
Examples of rule line thicknesses.

To change the thickness of an <hr> with CSS, use the `height` attribute, which I'll discuss in Lesson 8, "Using CSS to Style a Site."

The `width` attribute specifies the horizontal width of the rule line. You can specify the exact width of the rule in pixels. You can also specify the value as a percentage of the browser width (for example, 30% or 50%). If you set the width of a horizontal rule to a percentage, the width of the rule will change to conform to the window size if the user resizes the browser window. Alternatively, you can use the `width` CSS property instead. I'll also talk about width in the following lesson. Figure 7.8 shows the result of the following code, which displays some sample rule line widths:

Input ▼

```
<h2>100%</h2>
<hr />
<h2>75%</h2>
<hr width="75%" />
<h2>50%</h2>
<hr width="50%" />
<h2>25%</h2>
<hr width="25%" />
<h2>10%</h2>
<hr width="10%" />
```

Output ▶

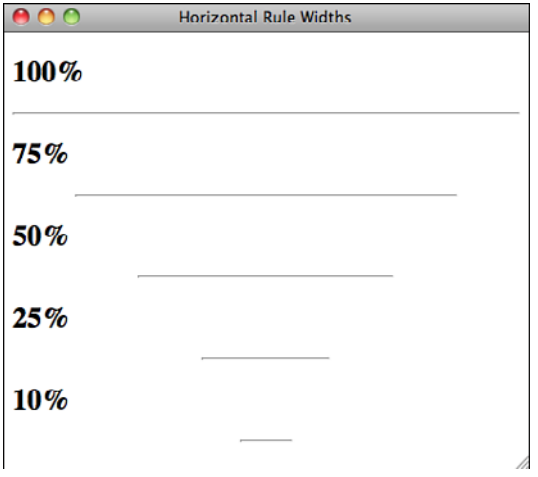


FIGURE 7.8
Examples of rule line widths.

If you specify a `width` smaller than the actual width of the browser window, you can also specify the alignment of that rule with the `align` attribute, making it flush left (`align="left"`), flush right (`align="right"`), or centered (`align="center"`). By default, rule lines are centered. The `align` attribute has been removed from HTML5 for all elements that once supported it. Alignment will be covered in the following lesson.

Finally, in most current browsers, the noshade attribute shown in the following example causes the browser to draw the rule line as a plain line without the three-dimensional shading, as shown in Figure 7.9.

Handling Attributes Without Values

In HTML 4.01 and, more recently, HTML5, a value isn't required by the noshade attribute. The method you use to apply this attribute appears as follows:

```
<hr align="center" size="4" width="200" noshade>
```

To comply with XHTML 1.0, however, all attributes require a value. The HTML 4.01 specification requires that Boolean attributes (such as noshade) have only the name of the attribute itself as the value. The following example demonstrates how to apply the noshade attribute to the <hr> tag in compliance with the XHTML 1.0 specification. To comply with HTML5, you would drop all of the attributes and use CSS instead. (The noshade attribute can be duplicated by changing the borders and background of the horizontal rule.)

```
<hr align="center" size="4" width="200" noshade="noshade" />
<hr align="center" size="4" width="300" noshade="noshade" />
<hr align="center" size="4" width="400" noshade="noshade" />
<h1 align="center">NorthWestern Video</h1>
<hr align="center" size="4" width="400" noshade="noshade" />
<hr align="center" size="4" width="300" noshade="noshade" />
<hr align="center" size="4" width="200" noshade="noshade" />
<h2 align="center">Presents</h2>
```

Output ▶

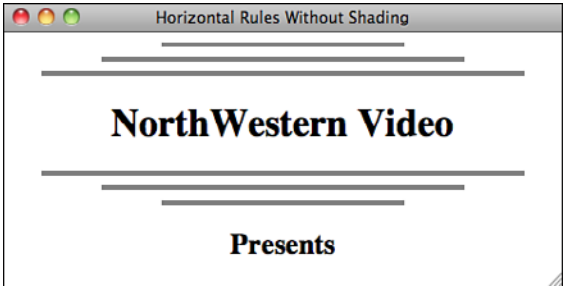


FIGURE 7.9
Rule lines without shading.

Line Break

The <br> tag breaks a line of text at the point where it appears. When a web browser encounters a <br> tag, it restarts the text after the tag at the left margin (whatever the current left margin happens to be for the current element). You can use <br> within other elements, such as paragraphs or list items;
 won't add extra space above or below the new line or change the font or style of the current entity. All it does is restart the text at the next line.

Closing Single Tags Properly

Like the <hr> tag, the <br> tag has no closing tag in HTML. To convert this tag to XHTML and to ensure compatibility with HTML browsers, add a space and forward slash to the end of the tag and its attributes, as shown in the following example:

```
And then is heard no more: it is a tale <br />
Told by an idiot, full of sound and fury, <br />
Signifying nothing.</p>
```

The following example shows a simple paragraph in which each line (except for the last, which ends with a closing <p> tag) ends with a <br>:

Input ▼

```
<p>Tomorrow, and tomorrow, and tomorrow,<br />
Creeps in this petty pace from day to day,<br />
To the last syllable of recorded time;<br />
And all our yesterdays have lighted fools<br />
The way to dusty death. Out, out, brief candle!<br />
Life's but a walking shadow; a poor player,<br />
That struts and frets his hour upon the stage,<br />
And then is heard no more: it is a tale <br />
Told by an idiot, full of sound and fury, <br />
Signifying nothing.</p>
```

Figure 7.10 shows how it appears in a browser.

NOTE

`clear` is an attribute of the <br> tag. It's used with images that have text wrapped alongside them. You'll learn about this attribute in Lesson 9, "Adding Images, Color, and Backgrounds."

Output ▶

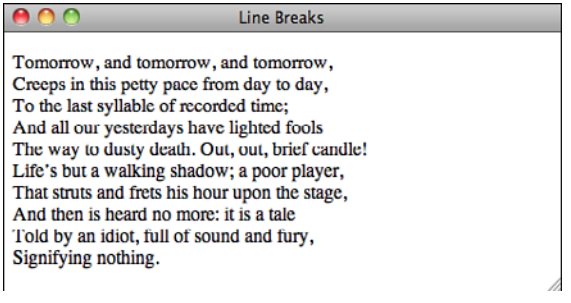


FIGURE 7.10
Line breaks.

Addresses

The address tag <address> is used for signature-like entities on web pages. Address tags usually go at the bottom of each web page and are used to indicate who wrote the web page, whom to contact for more information, the date, any copyright notices or other warnings, and anything else that seems appropriate. Addresses often are preceded with a rule line (<hr>), and the <br> tag can be used to separate the lines.

Without an address or some other method of signing your web pages, it's close to impossible to find out who wrote it or who to contact for more information. Signing each of your web pages using the <address> tag is an excellent way to make sure that people can get in touch with you. <address> is a block-level tag that italicizes the text inside it.

The following input shows an address:

Input ▼

```
<hr />
<address>
Laura Lemay <a href="mailto:lemay@lne.com">lemay@lne.com</a><br />
A service of Laura Lemay, Incorporated <br />
last revised January 10, 2010 <br />
Copyright Laura Lemay 2010 all rights reserved <br />
Void where prohibited. Keep hands and feet inside the vehicle at all times.
</address>
```

Figure 7.11 shows it in a browser.

Output ▶

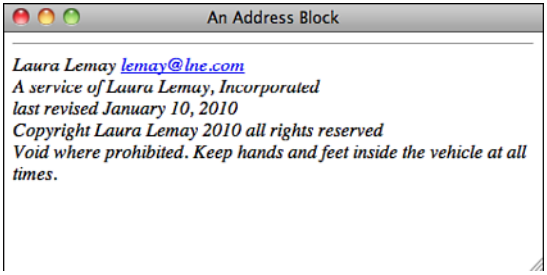


FIGURE 7.11
An address block.

Quotations

The <blockquote> tag is used to create an indented block of text within a page. (Unlike the <cite> tag, which highlights small quotes, <blockquote> is used for longer quotations that shouldn't be nested inside other paragraphs.) For example, the *Macbeth*

soliloquy I used in the example for line breaks would have worked better as a `<blockquote>` than as a simple paragraph. Here's an input example:

```
<blockquote>
"During the whole of a dull, dark, and soundless day in the autumn
of the year, when the clouds hung oppressively low in the heavens,
I had been passing alone, on horseback, through a singularly dreary
tract of country, and at length found myself, as the shades of evening
grew on, within view of the melancholy House of Usher."—-Edgar Allen Poe
</blockquote>
```

As with paragraphs, you can split lines in a `<blockquote>` using the line break tag, `<br>`. The following input example shows an example of this use:

Input ▼

```
<blockquote>
Guns aren't lawful, <br />
nooses give.<br />
gas smells awful.<br />
You might as well live.<br />
—-Dorothy Parker
</blockquote>
```

Figure 7.12 shows how the preceding input example appears in a browser.

Output ▶

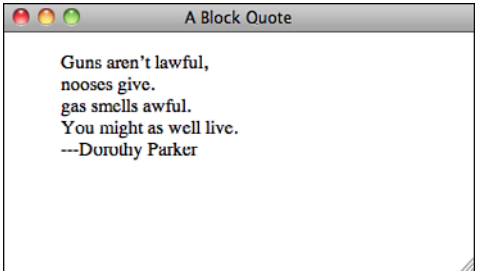


FIGURE 7.12
A block quotation.

NOTE

The `<blockquote>` tag is often used not to set off quotations within text, but rather to create margins on both sides of a page in order to make it more readable. This technique works, but strictly speaking, it's a misuse of the tag. These days, you should control margins with CSS, as explained in Lesson 8.

Special Characters

As you've already learned, HTML files are ASCII text and should contain no formatting or fancy characters. In fact, the only characters you should put in your HTML files are the characters that are actually printed on your keyboard. If you have to hold down any key other than Shift, or type an arcane combination of keys to produce a single character, you can't use that character in your HTML file. This includes characters you might use every day, such as em dashes and curly quotes. (If your word processor is set up to do automatic curly quotes, you should turn them off when you write your HTML files.)

"But wait a minute," you say. "If I can type a character like a bullet or an accented *a* on my keyboard using a special key sequence, and I can include it in an HTML file, and my browser can display it just fine when I look at that file, what's the problem?"

The problem is that the internal encoding your computer does to produce that character (which enables it to show up properly in your HTML file and in your browser's display) probably won't translate to other computers. Someone on the Internet who's reading your HTML file with that funny character in it might end up with some other character or just plain garbage.

So, what can you do? HTML provides a reasonable solution. It defines a special set of codes, called *character entities*, that you can include in your HTML files to represent the characters you want to use. When interpreted by a browser, these character entities display as the appropriate special characters for the given platform and font.

Some special characters don't come from the set of extended ASCII characters. For example, quotation marks and ampersands can be presented on a page using character entities even though they're found within the standard ASCII character set. These characters have a special meaning in HTML documents within certain contexts, so they can be represented with character entities to avoid confusing web browsers. Modern browsers generally don't have a problem with these characters, but it's not a bad idea to use the entities anyway.

CAUTION

HTML validators will complain when they encounter ampersands that are not part of entities, so you always want to encode them using entities on your pages.

Character Entities for Special Characters

Character entities take one of two forms: named entities and numbered entities.

Named entities begin with an ampersand (`&`) and end with a semicolon (`;`). In between is the name of the character (or, more likely, a shorthand version of that name, such as `agrave` for an *a* with a grave accent, or `reg` for a registered trademark sign). Unlike other HTML tags, the names are case-sensitive, so you should make sure to type them in exactly. Named entities look something like the following:

```
&agrave;
&quot;
&laquo;
&copy;
```

The numbered entities also begin with an ampersand and end with a semicolon, but rather than a name, they have a pound sign (`#`) and a number. The numbers correspond to character positions in the ISO-Latin-1 (ISO 8859-1) character set. Every character you can type or for which you can use a named entity also has a numbered entity. Numbered entities look like the following:

```
&#130;
&#245;
```

You can use either numbers or named entities in your HTML file by including them in the same place that the character they represent would go. So, to place the word *résumé* in your HTML file, you would use either

```
r&eacute;sum&eacute;
```

or

```
r&#233;sum&#233;
```

Appendix B, "HTML Quick Reference," includes a table that lists the named entities currently supported by HTML. See that table for specific characters.

Character Sets

All web pages are rendered using a character set. A character set represents all the letters, numbers, and symbols that can be displayed on the screen. HTML originally supported only the ISO-Latin-1 character set, which supports most accented characters, but not common characters such as bullets, em dashes, and curly quotes. Usually when you see a page that includes nonsensical symbols where punctuation should be, it's because the browser is using a different character set than the one that was used to create the document.

HTML 4.01 took things a huge leap further by adding support for Unicode. Unicode is a standard character encoding system that, although backward compatible with our familiar ASCII encoding, offers the capability to encode characters in almost any of the world's languages, including Chinese and Japanese. This means that documents can support any language, and that one document can contain multiple languages. All modern browsers support Unicode, and can render documents that use all the characters provided by Unicode as long as the necessary fonts are available.

Specifying Character Encoding

Most characters on a web page are the same in Unicode (UTF-8), Western (ISO-8859-1), and Western (Windows-1252). However, some more exotic characters differ between them. If these characters are included on a web page and the character encoding does not include those characters, they will display incorrectly. For example, if the browser thinks a page is encoded in UTF-8 and the page includes smart (curly) quotation marks that were copied from a Word document, those characters will display as unintelligible symbols.

This is one of the reasons to use named entities wherever possible. Browsers will translate those entities to the proper characters, regardless of the character set used. You can change the character set the browser is using for a web page. This can prove useful when a page has the wrong encoding, or if you just want to experiment with different character sets. In my browser, you can change character sets using the Character Encoding item in the View menu. The character encoding used for the page will be selected in that menu, and you can change to another one by selecting it.

You can also specify the character set used by a page in the page header, using a `<meta>` tag. Here's a tag that will specify UTF-8 as the character encoding for that page:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

This indicates to the browser that the page is an HTML page encoded using `utf-8`. The `<meta>` must be placed somewhere within the `<head>` tag. The `<meta>` tag is used to specify or override information that is normally provided by the web server, and indeed, web server software enables you to specify a character encoding for all the pages that it serves. If you need to use a specific character encoding for all the pages on your site, you'll want to configure your web server to use that encoding, instead of adding a `<meta>` tag to every page on your site.

Character Entities for Reserved Characters

For the most part, character entities exist so that you can include special characters that aren't part of the standard ASCII character set. However, there are several exceptions for the few characters that have special meaning in HTML itself. You must use entities for these characters, too.

Suppose that you want to include a line of code that looks something like the following in an HTML file:

```
<p><code>if x < 0 do print i</code></p>
```

Doesn't look unusual, does it? Unfortunately, HTML cannot display this line as written. Why? The problem is with the < (less-than) character. To an HTML browser, the less-than character means "this is the start of a tag." Because the less-than character isn't actually the start of a tag in this context, your browser might get confused. You'll have the same problem with the greater-than character (>) because it means the end of a tag in HTML, and with the ampersand (&) because it signals the beginning of an entity. Written correctly for HTML, the preceding line of code would look like the following instead:

```
<p><code>if x &lt; 0 do print i</code></p>
```

HTML provides named entities for each of these characters, and one for the double quotation mark, too, as shown in Table 7.1.

TABLE 7.1 Escape Codes for Characters Used by Tags

Entity	Result
<	<
>	>
&	&
"	"

The double quotation mark escape is the mysterious one. Technically, if you want to include a double quotation mark in text, you should use the escape sequence, and you shouldn't type the quotation mark character. However, I haven't noticed any browsers having problems displaying the double quotation mark character when it's typed literally in an HTML file, nor have I seen many HTML files that use it. For the most part, you're probably safe using plain old quotes (") in your HTML files rather than the escape code. Furthermore, HTML validators will not flag quotation marks that are not encoded using entities.

Text Alignment

Text alignment is the capability to arrange a block of text, such as a heading or a paragraph, so that it's aligned against the left margin (left justification, the default), aligned against the right margin (right justification), or centered. The old-fashioned approach to aligning elements is to use the `align` attribute, which was deprecated as of HTML 4.01 and has been removed completely from HTML5. The approach currently in favor is to use CSS, as explained next.

Aligning Individual Elements

To align an individual heading or paragraph, include the `align` attribute in the opening tag. The `align` attribute has four values: `left`, `right`, `center`, and `justify`. Consider the following examples in the code snippet that follows.

The following input and output example shows the simple alignment of several headings. Figure 7.13 shows the results.

Input ▼

```
<h1 align="center">Northridge Paints, Inc.</h1>
<p align="center">We don't just paint the town red.</p>

<h1 align="left">Serendipity Products</h1>
<h2 align="right"><a href="who.html">Who We Are</a></h2>
<h2 align="right"><a href="products.html">What We Do</a></h2>
<h2 align="right"><a href="contacts.html">How To Reach Us</a></h2>
```

Output ▶

FIGURE 7.13
Headings with varying alignments.

Aligning Blocks of Elements

A slightly more flexible method of aligning text elements is to use the `<div>` tag. The `<div>` tag supports several attributes. Among these attributes is `align` (deprecated as of HTML 4.01), which aligns elements to the left, right, or center just as it does for

headings and paragraphs. Unlike using alignments in individual elements, however, is used to surround a block of HTML tags of any kind, and it affects all the tags and text inside the opening and closing tags. Two advantages of div over the align attribute follow:

- You need to use only once, rather than including align repeatedly in several different tags.
- You can use to align anything (headings, paragraphs, quotes, images, tables, and so on); the align attribute is available on only a limited number of tags.

To align a block of HTML code, surround it with opening and closing tags, and then include the align attribute in the opening tag. As in other tags, align can have the value left, right, or center:

```
<h1 align="left">Serendipity Products</h1>
<div align="right">
<h2><a href="who.html">Who We Are</a></h2>
<h2><a href="products.html">What We Do</a></h2>
<h2><a href="contacts.html">How To Reach Us</a></h2>
</div>
```

All the HTML between the two tags will be aligned according to the value of the align attribute. If individual align attributes appear in headings or paragraphs inside the , those values will override the global setting.

Note that itself isn't a paragraph type; it's just a container. Rather than altering the layout of the text itself, it just enables you to apply your own styles to the text and tags inside. One function of is to change text alignment with the align attribute. It's also often used with CSS to apply styles to a specific block of text (much like its counterpart,). In fact, to center elements within the the CSS way (instead of using the deprecated align attribute), you can use the text-align property. Valid values for it are left, right, center, and justify. Figure 7.14 shows how it's used.

Input ▼

```
<div style="text-align: left">Left aligned text.</div>
<div style="text-align: right">Right aligned text.</div>
<div style="text-align: center">Centered text.</div>
<div style="text-align: justify">This text is justified. I'm adding some extra
text for padding so that you can see exactly how the justification works. As you
can see, the text is expanded so that it is aligned with both the left and right
margins.</div>
```

Output ▶

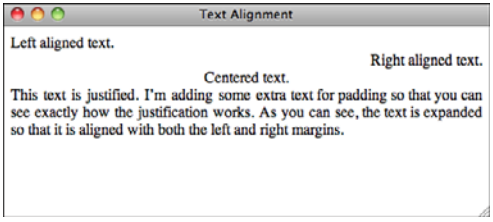


FIGURE 7.14
Various text alignments available using CSS.

You can also include the `align` attribute in the `<p>` tag. It's most common to use the `justify` setting for the `align` attribute with the `<p>` and `<div>` tags. When you justify a paragraph, the text is spaced so that it's flush with both the left and right margins of the page.

Fonts and Font Sizes

Earlier in this lesson, I described a few font-related properties that you can manipulate using CSS. In fact, you can use CSS to control all font usage on the page. I also described how the `font-family` property can be used to specify that text should be rendered in a font belonging to a particular general category, such as monospace or serif. You can also use the `font-family` property to specify a specific font.

You can provide a single font or a list of fonts, and the browser will search for each of the fonts until it finds one on your system that appears in the list. You can also include a generic font family in the list of fonts if you like, just as you can with the `<font>` tag. Here are some examples:

```
<p style="font-family: Verdana, Trebuchet, Arial, sans-serif">
This is sans-serif text.</p>
<p style="font-family: Courier New, monospace">This is
monospace text.</p>
<p style="font-family: Georgia">This text will appear in the
Georgia font, or, if that font is not installed, the browser's
default font.</p>
```

You can also use CSS to specify font size. Unfortunately, although the approach for specifying the font face itself is the same whether you're using the `<font>` tag or CSS, specifying font sizes under CSS is much more complicated than it is with the `<font>` tag. The trade-off is that with this complexity comes a great degree more flexibility in how font sizes can be specified. Let's start with the basics. To change the font size for some text, the `font-size` property is used. The value is a size (relative or absolute) in any of the units of measure supported by CSS.

The catch here is that several units of measure are available. Perhaps the simplest is the percentage size, relative to the current font size being used. So, to make the font twice as large as it is currently, just use the following:

```
<p>This text is normal sized, and this text is
<span style="font-size: 200%">twice that size</span>.</p>
```

A number of length units are also available that you can use to specify the font size in absolute terms. I discuss the popular ones in Lesson 10. In the meantime, just know that there are two kinds of length units: *relative units* and *absolute units*. Relative units are sized based on the size of other elements on the page and based on the dots per inch setting of the user's display. Absolute units are sized based on some absolute reference. For example, the `pt` (point) unit is measured in absolute pixels. To set your text to be exactly 12 pixels high, the following specification is used:

```
<p style="font-size: 12px">This text is 12 pixels tall.</p>
```

CAUTION

One thing to watch out for: When you specify units in CSS, you must leave no spaces between the number of units and unit specification. In other words, `12pt` and `100%` are valid, and `12 pt` and `100 %` aren't.

You can do another thing with the `font-size` property that's not possible with the `<font>` tag: specify line height. Let's say you want to use double-spaced text on your page. Before CSS, the only way to achieve the effect was to use the `<br>` tag inside paragraphs to skip lines, but this approach is fraught with peril. Depending on how the user has sized her browser window, pages formatted using `<br>` in this manner can look truly awful. To set the line height using CSS, you can include it in your font size specification, like this: `font-size: 100%/200%`. In this case, the size of the font is `100%`—the default—and the line height is `200%`, twice the standard line height.

DO	DON'T
Do list backup fonts when specifying a font family to make it more likely that your users will have one of the fonts you specify.	**Don't** use too many different fonts on the same page.
Do take advantage of the ability to change the line height to improve readability.	**Don't** use absolute font sizes with CSS if you can help it, because some browsers won't let users alter the text size if you do so.

Task: Exercise 7.1: Creating a Real HTML Page

Here's your chance to apply what you've learned and create a real web page. No more disjointed or overly silly examples. The web page you create in this section is a real one, suitable for use in the real world (or the real world of the web, at least).

Your task for this example is to design and create a home page for a bookstore called The Bookworm, which specializes in old and rare books.

Planning the Page Lesson 2, "Preparing to Publish on the Web," mentioned that planning your web page before writing it usually makes building and maintaining the elements easier. First, consider the content you want to include on this page. The following are some ideas for topics for this page:

- The address and phone number of the bookstore
- A short description of the bookstore and why it's unique
- Recent titles and authors
- Upcoming events

Now come up with some ideas for the content you're going to link to from this page. Each title in a list of recently acquired books seems like a logical candidate. You also can create links to more information about each book, its author and publisher, its price, and maybe even its availability.

The Upcoming Events section might suggest a potential series of links, depending on how much you want to say about each event. If you have only a sentence or two about each one, describing them on this page might make more sense than linking them to another page. Why make your readers wait for each new page to load for just a couple of lines of text?

Other interesting links might arise in the text itself, but for now, starting with the basic link plan is enough.

Beginning with a Framework Next, create the framework that all HTML files must include: the document structure, a title, and some initial headings. Note that the title is descriptive but short; you can save the longer title for the `<h1>` element in the body of the text. The four `<h2>` subheadings help you define the four main sections you'll have on your web page:

```
<!DOCTYPE html><html>
<head>
<title>The Bookworm Bookshop</title>
</head>
<body>
```

```
<h1>The Bookworm: A Better Book Store</h1>
<h2>Contents</h2>
<h2>About the Bookworm Bookshop</h2>
<h2>Recent Titles (as of 11-Jan-2010)</h2>
<h2>Upcoming Events</h2>
</body>
</html>
```

Each heading you've placed on your page marks the beginning of a particular section. You'll create an anchor at each of the topic headings so that you can jump from section to section with ease. The anchor names are simple: `top` for the main heading; `contents` for the table of contents; and `about`, `recent`, and `upcoming` for the three subsections on the page. With the anchors in place, the revised code looks like the following:

Input ▼

```
<!DOCTYPE html><html>
<head>
<title>The Bookworm Bookshop</title>
</head>
<body>
<a name="top"><h1>The Bookworm: A Better Book Store</h1></a>
<a name="contents"><h2>Contents</h2></a>
<a name="about"><h2>About the Bookworm Bookshop</h2></a>
<a name="recent"><h2>Recent Titles (as of 11-Jan-2010)</h2></a>
<a name="upcoming"><h2>Upcoming Events</h2></a>
</body>
</html>
```

Adding Content Now begin adding the content. You're undertaking a literary endeavor, so starting the page with a nice quote about old books would be a nice touch. Because you're adding a quote, you can use the <blockquote> tag to make it stand out as such. Also, the name of the poem is a citation, so use <cite> there, too.

Insert the following code on the line after the level 1 heading:

Input ▼

```
<blockquote>
"Old books are best—how tale and rhyme<br />
Float with us down the stream of time!"<br />
- Clarence Urmy, <cite>Old Songs are Best</cite>
</blockquote>
```

Immediately following the quote, add the address for the bookstore. This is a simple paragraph with the lines separated by line breaks, like the following:

Input

```
<p>The Bookworm Bookshop<br />
1345 Applewood Dr<br />
Springfield, CA 94325<br />
(415) 555-0034
</p>
```

Adding the Table of Contents The page you're creating will require a lot of scrolling to get from the top to the bottom. One nice enhancement is to add a small table of contents at the beginning of the page, listing the sections in a bulleted list. If a reader clicks one of the links in the table of contents, he'll automatically jump to the section that's of most interest to him. Because you've created the anchors already, it's easy to see where the links will take you.

You already have the heading for the table of contents. You just need to add the bulleted list and a horizontal rule, and then create the links to the other sections on the page. The code looks like the following:

Input

```
<a name="contents"><h2>Contents</h2></a>
<ul>
 <li><a href="#about">About the Bookworm Bookshop</a></li>
 <li><a href ="#recent">Recent Titles</a></li>
 <li><a href ="#upcoming">Upcoming Events</a></li>
</ul>
<hr />
```

Figure 7.15 shows an example of the introductory portion of the Bookworm Bookshop page as it appears in a browser.

Output

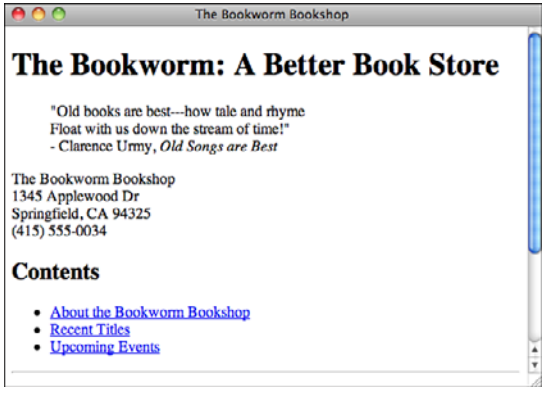


FIGURE 7.15
The top section of the Bookworm Bookshop page.

Creating the Description of the Bookstore Now you come to the first descriptive subheading on the page, which you've added already. This section gives a description of the bookstore. After the heading (shown in the first line of the following example), I've arranged the description to include a list of features to make them stand out from the text better:

Input

```
<a name="about"><h2>About the Bookworm Bookshop</h2></a>
<p>Since 1933, The Bookworm Bookshop has offered
rare and hard-to-find titles for the discerning reader.
The Bookworm offers:</p>
<ul>
<li>Friendly, knowledgeable, and courteous help</li>
<li>Free coffee and juice for our customers</li>
<li>A well-lit reading room so you can "try before you buy"</li>
<li>Four friendly cats: Esmerelda, Catherine, Dulcinea and Beatrice</li>
</ul>
```

Add a note about the hours the store is open and emphasize the actual numbers:

Input

```
<p>Our hours are <strong>10am to 9pm</strong> weekdays,
<strong>noon to 7</strong> on weekends.</p>
```

Then, end the section with links to the table of contents and the top of the page, followed by a horizontal rule to end the section:

Input

```
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to Top</a></p>
<hr />
```

Figure 7.16 shows you what the About the Bookworm Bookshop section looks like in a browser.

Output ▶

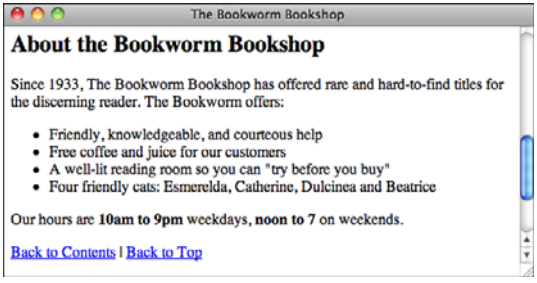


FIGURE 7.16
The About the Bookworm Bookshop section.

Creating the Recent Titles Section The Recent Titles section is a classic link menu, as described earlier in this section. Here you can put the list of titles in an unordered list, with the titles as citations, by using the `<cite>` tag. End the section with another horizontal rule.

After the Recent Titles heading (shown in the first line in the following example), enter the following code:

```
<a name="recent"><h2>Recent Titles (as of 11-Jan-2010)</h2></a>
<ul>
<li>Sandra Bellweather, <cite>Belladonna</cite></li>
<li>Jonathan Tin, <cite>20-Minute Meals for One</cite></li>
<li>Maxwell Burgess, <cite>Legion of Thunder</cite></li>
<li>Alison Caine, <cite>Banquo's Ghost</cite></li>
</ul>
<hr />
```

Now add the anchor tags to create the links. How far should the link extend? Should it include the whole line (author and title) or just the title of the book? This decision is a matter of preference, but I like to link only as much as necessary to make sure that the link stands out from the text. I prefer this approach to overwhelming the text. Here, I linked only the titles of the books. At the same time, I also added links to the table of contents and the top of the page:

Input ▼

```
<a name="recent"><h2>Recent Titles (as of 11-Jan-2010)</h2></a>
<ul>
<li>Sandra Bellweather, <a href="belladonna.html">
<cite>Belladonna</cite></a></li>
<li>Johnathan Tin, <a href="20minmeals.html">
<cite>20-Minute Meals for One</cite></a></li>
<li>Maxwell Burgess, <a href="legion.html">
<cite>Legion of Thunder</cite></a></li>
<li>Alison Caine, <a href="banquo.html">
<cite>Banquo's Ghost</cite></a></li>
</ul>
```

```
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to Top</a></p>
<hr />
```

Note that I put the <cite> tag inside the link tag <a>. I could have just as easily put it outside the anchor tag; character style tags can go just about anywhere. But as mentioned before, be careful not to overlap tags. Your browser might not understand what's going on, and it's invalid. In other words, don't do the following:

```
<a href="banquo.html"><cite>Banquo's Ghost</a></cite>
```

Take a look at how the Recent Titles section appears in Figure 7.17.

Output ▶

FIGURE 7.17
The Recent Titles section.

Completing the Upcoming Events Section Next, move on to the Upcoming Events section. In the planning stages, you weren't sure whether this would be another link menu or whether the content would work better solely on this page. Again, this decision is a matter of preference. Here, because the amount of extra information is minimal, creating links for just a couple of sentences doesn't make much sense. So, for this section, create an unordered list using the <ul> tag. I've boldfaced a few phrases near the beginning of each paragraph. These phrases emphasize a summary of the event so that the text can be scanned quickly and ignored if the readers aren't interested.

As in the previous sections, you end the section with links to the top and to the table of contents, followed by a horizontal rule:

```
 <a name="upcoming"><h2>Upcoming Events</h2></a>
<ul>
<li><b>The Wednesday Evening Book Review</b> meets, appropriately, on
Wednesday evenings at 7 pm for coffee and a round-table discussion.
Call the Bookworm for information on joining the group.</li>
<li><b>The Children's Hour</b> happens every Saturday at 1 pm and includes
reading, games, and other activities. Cookies and milk are served.</li>
<li><b>Carole Fenney</b> will be at the Bookworm on Sunday, January 19,
to read from her book of poems <cite>Spiders in the Web.</cite></li>
<li><b>The Bookworm will be closed</b> March 1st to remove a family
of bats that has nested in the tower. We like the company, but not
the mess they leave behind!</li>
```

```
</ul>
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to
Top</a></p>
```

Signing the Page To finish, sign what you have so that your readers know who did the work. Here, I've separated the signature from the text with a rule line. I've also included the most recent revision date, my name as the webmaster, and a basic copyright (with a copyright symbol indicated by the numeric escape `©`):

Input ▼

```
<hr />
<address>
Last Updated: 11-Jan-2010<br />
Webmaster: Laura Lemay
<a href="mailto:lemay@bookworm.com">lemay@bookworm.com</a><br />
&#169; copyright 2010 the Bookworm<br />
</address>
```

Figure 7.18 shows the signature at the bottom portion of the page and the Upcoming Events section.

Output ▶

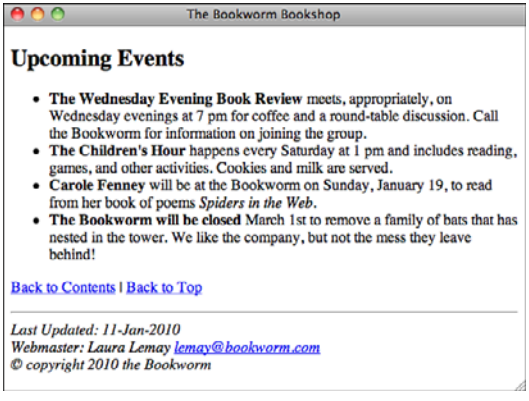


FIGURE 7.18
The Upcoming Events section and the page signature.

Reviewing What You've Got Here's the HTML code for the page so far:

```
<!DOCTYPE html><html>
<head>
<title>The Bookworm Bookshop</title>
</head>
<body>
<a name="top"><h1>The Bookworm: A Better Book Store</h1></a>
<blockquote>
"Old books are best—how tale and rhyme<br />
Float with us down the stream of time!"<br />
```

```
- Clarence Urmy, <cite>Old Songs are Best</cite>
</blockquote>
<p>The Bookworm Bookshop<br />
1345 Applewood Dr<br />
Springfield, CA 94325<br />
(415) 555-0034
</p>
<a name="contents"><h2>Contents</h2></a>
<ul>
 <li><a href="#about">About the Bookworm Bookshop</a></li>
 <li><a href="#recent">Recent Titles</a></li>
 <li><a href="#upcoming">Upcoming Events</a></li>
</ul>
<hr />
<a name="about"><h2>About the Bookworm Bookshop</h2></a>
<p>Since 1933, the Bookworm Bookshop has offered
rare and hard-to-find titles for the discerning reader.
The Bookworm offers:</p>
<ul>
 <li>Friendly, knowledgeable, and courteous help</li>
 <li>Free coffee and juice for our customers</li>
 <li>A well-lit reading room so you can "try before you buy"</li>
 <li>Four friendly cats: Esmerelda, Catherine, Dulcinea and Beatrice</li>
</ul>
<p>Our hours are <strong>10am to 9pm</strong> weekdays,
<strong>noon to 7</strong> on weekends.</p>
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to
Top</a></p>
<hr />
<a name="recent"><h2>Recent Titles (as of 11-Jan-2010)</h2></a>
<ul>
 <li>Sandra Bellweather, <a href="belladonna.html">
  <cite>Belladonna</cite></a></li>
 <li>Johnathan Tin, <a href="20minmeals.html">
  <cite>20-Minute Meals for One</cite></a></li>
 <li>Maxwell Burgess, <a href="legion.html">
  <cite>Legion of Thunder</cite></a></li>
 <li>Alison Caine, <a href="banquo.html">
  <cite>Banquo's Ghost</cite></a></li>
</ul>
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to
Top</a></p>
<hr />
<a name="upcoming"><h2>Upcoming Events</h2></a>
<ul>
 <li><b>The Wednesday Evening Book Review</b> meets, appropriately, on
   Wednesday evenings at 7 pm for coffee and a round-table discussion.
   Call the Bookworm for information on joining the group.</li>
 <li><b>The Children's Hour</b> happens every Saturday at 1 pm and includes
   reading, games, and other activities. Cookies and milk are served.</li>
 <li><b>Carole Fenney</b> will be at the Bookworm on Sunday, January 19,
```

```
   to read from her book of poems <cite>Spiders in the Web.</cite></li>
 <li><b>The Bookworm will be closed</b> March 1 to remove a family
   of bats that has nested in the tower. We like the company, but not
   the mess they leave behind!</li>
</ul>
<p><a href="#contents">Back to Contents</a> ¦ <a href="#top">Back to
Top</a></p>
<hr />
<address>
Last Updated: 11-Jan-2010<br />
WebMaster: Laura Lemay lemay@bookworm.com<br />
&#169; copyright 2010 the Bookworm<br />
</address>
</body>
</html>
```

Now you have some headings, some text, some topics, and some links, which form the basis for an excellent web page. With most of the content in place, now you need to consider what other links you might want to create or what other features you might want to add to this page.

For example, the introductory section has a note about the four cats owned by the bookstore. Although you didn't plan for them in the original organization, you could easily create web pages describing each cat (and showing pictures) and then link them back to this page, one link (and one page) per cat.

Is describing the cats important? As the designer of the page, that's up to you to decide. You could link all kinds of things from this page if you have interesting reasons to link them (and something to link to). Link the bookstore's address to an online mapping service so that people can get driving directions. Link the quote to an online encyclopedia of quotes. Link the note about free coffee to the Coffee home page.

I cover more good things to link (and how not to get carried away when you link) in Lesson 18, "Writing Good Web Pages: Do's and Don'ts." My reason for bringing up this point here is that after you have some content in place on your web pages, there might be opportunities for extending the pages and linking to other places that you didn't think of when you created your original plan. So, when you're just about finished with a page, stop and review what you have, both in the plan and on your web page.

For the purposes of this example, stop here and stick with the links you have. You're close enough to being done, and I don't want to make this lesson any longer than it already is!

Testing the Result Now that all the code is in place, you can preview the results in a browser. Figures 7.16 through 7.19 show how it looks in a browser. Actually, these figures show what the page looks like after you fix the spelling errors, the forgotten closing

tags, and all the other strange bugs that always seem to creep into an HTML file the first time you create it. These problems always seem to happen no matter how good you are at creating web pages. If you use an HTML editor or some other help tool, your job will be easier, but you'll always seem to find mistakes. That's what previewing is for—so you can catch the problems before you actually make the document available to other people.

Getting Fancy Everything included on the page up to this point has been plain-vanilla HTML 2.0, so it's readable and will look pretty much the same in all browsers. After you get the page to this point, however, you can add additional formatting tags and attributes that won't change the page for many readers but might make it look a little fancier in browsers that do support these attributes.

So, what attributes do you want to use? Here are two:

- Centering the title of the page, the quote, and the bookstore's address
- Making a slight font size change to the address

To center the topmost part of the page, you can use the `<div>` tag around the heading, the quote, and the bookshop's address, as in the following:

Input

```
<div style="text-align: center">
<a name="top"><h1 style="font-variant: small-caps">The Bookworm: A Better Book
Store</h1></a>
<blockquote>
"Old books are best—how tale and rhyme<br />
Float with us down the stream of time!"<br />
- Clarence Urmy, <cite>Old Songs are Best</cite>
</blockquote>
<p>The Bookworm Bookshop<br />
1345 Applewood Dr<br />
Springfield, CA 94325<br />
(415) 555-0034
</p>
</div>
```

I've also used the `style` attribute to change the text in the `<h1>` tag to small caps. To change the font size of the address, add a `style` attribute to the paragraph containing the address:

Input ▼

```
<p style="font-size: 150%">The Bookworm Bookshop<br />
1345 Applewood Dr<br />
Springfield, CA 94325<br />
(415) 555-0034
</p>
```

Figure 7.19 shows the final result, including styles.

Output ▶

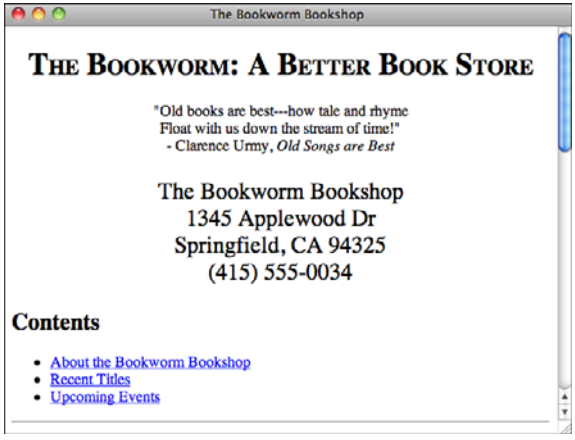


FIGURE 7.19
The final Bookworm home page, with additional attributes.

You'll learn more about formatting tags, and styles, and how to design well with them, in Lesson 8. ▲

Summary

Tags, tags, and more tags! In this lesson, you learned about most of the remaining tags in the HTML language for presenting text, and quite a few of the tags for additional text formatting and presentation. You also put together a real-life HTML home page. You could stop now and create quite presentable web pages, but more cool stuff is to come. So, don't put down the book yet.

Table 7.2 presents a quick summary of all the tags and attributes you've learned about in this lesson. Table 7.3 summarizes the CSS properties that have been described in this lesson.

TABLE 7.2 HTML Tags from Lesson 7

Tag	Attribute	Use
`<address>...</address>`		A signature for each web page; typically occurs near the bottom of each document and contains contact or copyright information.
`<b>...</b>`		Bold text.
`<big>...</big>`		Text in a larger font than the text around it. (Removed from HTML5.)
`<blockquote>...</blockquote>`		A quotation longer than a few words.
`<cite>...</cite>`		A citation.
`<code>...</code>`		A code sample.
`<dfn>...</dfn>`		A definition, or a term about to be defined.
`<em>...</em>`		Emphasized text.
`<i>...</i>`		Italic text.
`<kbd>...</kbd>`		Text to be typed in by the user.
`<pre>...</pre>`		Preformatted text; all spaces, tabs, and returns are retained. Text is printed in a monospaced font.
`<s>...</s>` from HTML5)		Strikethrough text. (Removed
`<samp>...</samp>`		Sample text.
`<small>...</small>`		Text in a smaller font than the text around it.
`<strong>...</strong>`		Strongly emphasized text.
`<sub>...</sub>`		Subscript text.
`<sup>...</sup>`		Superscript text.
`<tt>...</tt>`		Text in typewriter font (a monospaced font such as Courier). (Removed from HTML5.)
`<u>...</u>`		Underlined text. (Removed from HTML5.)
`<var>...</var>`		A variable name.
`<span>...</span>`		A generic tag used to apply styles to a particular bit of text.

TABLE 7.2 Continued

Tag	Attribute	Use
<hr>		A horizontal rule line at the given position in the text. There's no closing tag in HTML for <hr>; for XHTML, add a space and forward slash (/) at the end of the tag and its attributes (for example, <hr size="2" width="75%" />).
	size	The thickness of the rule, in pixels. (Removed from HTML5.)
	width	The width of the rule, either in exact pixels or as a percentage of page width (for example, 50%). (Removed from HTML5.)
	align	The alignment of the rule on the page. Possible values are left, right, and center. (Removed from HTML5.)
	noshade	Displays the rule without three-dimensional shading. (Removed from HTML5.
 		A line break; starts the next character on the next line but doesn't create a new paragraph or list item. There's no closing tag in HTML for ; for XHTML, add a space and forward slash (/) at the end of the tag and its attributes (for example, <br clear="left" />).
<nobr>...</nobr>		Doesn't wrap the enclosed text (nonstandard; supported by Netscape and Internet Explorer).
<wbr>		Wraps the text at this point only if necessary (nonstandard; supported by Netscape and Internet Explorer). Adds a space and forward slash at the end of the tag for XHTML 1.0.
... , <h1-6>...</h1-6>	align="left"	Left-justifies the text within that paragraph or heading. (Removed from HTML5.)

TABLE 7.2 Continued

Tag	Attribute	Use
	`align="right"`	Right-justifies the text within that paragraph or heading. (Removed from HTML5.)
	`align="center"`	Centers the text within that paragraph or heading. (Removed from HTML5.)
`<div>...</div>`	`align="left"`	Left-justifies all the content between the opening and closing tags. (DeprecatRemoved from HTML5.)
	`align="right"`	Right-justifies all the content between the opening and closing tags. (Removed from HTML5.)
	`align="center"`	Centers all the content between the opening and closing tags. (Removed from HTML5.)
`<center>...</center>`		Centers all the content between the opening and closing tags. (Removed from HTML5.)

TABLE 7.3 CSS Properties from Lesson 7

Property	Use/Values
`text-decoration`	Specifies which sort of decoration should be applied to the text. The values are `underline`, `overline`, `line-through`, `blink`, and `none`.
`font-style`	Specifies whether text should be italicized. The three values are `normal`, `italic`, and `oblique`.
`font-weight`	Specifies the degree to which text should be emboldened. Options are `normal`, `bold`, `bolder`, `lighter`, and `100` to `900`.
`font-family`	Enables you to specify the font used for text. You can choose families such as serif, sans serif, and monospace, or specific font names. You can also specify more than one font or font family.
`font-variant`	Sets the font variant to `normal` or `small-caps`.
`text-align`	Specifies how text is aligned: `left`, `right`, `center`, or `justify`.
`font-size`	Enables you to specify the font size in any unit supported by CSS.

Workshop

Here you are at the close of this lesson (a long one!) and facing yet another workshop. This lesson covered a lot of ground, so I'll try to keep the questions easy. There are a couple exercises that focus on building some additional pages for your website. Ready?

Q&A

Q If line breaks appear in HTML, can I also do page breaks?

A HTML doesn't have a page break tag. Consider what the term *page* means in a web document. If each document on the web is a single page, the only way to produce a page break is to split your HTML document into separate files and link them.

Even within a single document, browsers have no concept of a page; each HTML document simply scrolls by continuously. If you consider a single screen a page, you still can't have what results in a page break in HTML. The screen size in each browser is different. It's based on not only the browser itself, but also the size of the monitor on which it runs, the number of lines defined, the font currently being used, and other factors that you cannot control from HTML.

When you design your web pages, don't get too hung up on the concept of a page the way it exists in paper documents. Remember, HTML's strength is its flexibility for multiple kinds of systems and formats. Instead, think in terms of creating small chunks of information and how they link together to form a complete presentation.

If page breaks are essential to your document, you might consider saving it in the PDF format and making it available for download.

Q How can I include em dashes or curly quotes (typesetter's quotes) in my HTML files?

A There are entities for all these characters, but they might not be supported by all browsers or on all platforms. Most people still don't use them. To add an em dash, use `—`. The curly quote entities are `“` for the left quote and `”` for the right quote. Similarly, you can create curly single quotes using `‘` and `’`.

Quiz

1. What are the differences between logical character styles and physical character styles?
2. What are some things that the <pre> (preformatted text) tag can be used for?
3. What's the most common use of the <address> tag?
4. Older versions of HTML provided ways to align and center text on a web page. What's the recommended way to accomplish these tasks in HTML 4.01?
5. Without looking at Table 7.2, list all eight logical style tags and what they're used for. Explain why you should use the logical tags rather than the physical tags.

Quiz Answers

1. Logical styles indicate how the highlighted text is used (citation, definition, code, and so on). Physical styles indicate how the highlighted text is displayed (bold, italic, or monospaced, for example).
2. Preformatted text can be used for text-based tables, code examples, ASCII art, and any other web page content that requires extra spaces to align characters.
3. The <address> tag is most commonly used for signature-like entities on a web page. These include the name of the author of the web page, contact information, dates, copyright notices, or warnings. Address information usually appears at the bottom of a web page.
4. Alignment and centering of text can be accomplished with style sheets, which is the recommended approach in HTML 4.01.
5. The eight logical styles are (for emphasized text), (for bold text), <code> (for programming code), <samp> (similar to <code>), <kbd> (to indicate user keyboard input), <var> (for variable names), <dfn> (for definitions), and <cite> (for short quotes or citations). Logical tags rely on the browser to format their appearance.

Exercises

1. Now that you've had a taste of building your first thorough web page, take a stab at your own home page. What can you include that would entice people to dig deeper into your pages? Don't forget to include links to other pages on your site.
2. Try out your home page in several browsers and even on multiple platforms if you have access to them. Web developers have to get used to the fact that their designs are at the mercy of their users, and it's best to see right away how different browsers and platforms treat pages.

LESSON 8
Using CSS to Style a Site

The past few lessons Idiscussed how to lay out web pages using *Hypertext Markup Language (HTML)* tags. This lesson describes how you can create complex pages using Cascading Style Sheets (CSS). You've already learned about the advantages CSS can provide for formatting smaller snippets of text. In this lesson, you learn how to use CSS to control the appearance of an entire page.

The following topics are covered:

- Creating style sheets and including them in a page
- Linking to external style sheets
- Using selectors to apply styles to elements on a page
- Units of measure supported by CSS
- The CSS box model
- Positioning elements using CSS
- Applying styles to tables and the `<body>` tag
- Using CSS to create multicolumn layouts

Including Style Sheets in a Page

Thus far, when I've discussed style sheets, I've applied them using the `style` attribute of tags. For example, I've shown how you can modify the font for some text using tags such as `<div>` and `<span>`, or how you can modify the appearance of a list item by applying a style within an `<li>` tag. If you rely on the `style` attribute of tags to apply CSS, if you want to embolden every paragraph on a page, you need to put `style="font-weight: bold"` in every `<p>` tag. This is no improvement over just using `<p><b>` and `</b></p>` instead. Fortunately, CSS provides ways to apply styles generally to a page, or even to an entire website.

Creating Page-Level Styles

First, let's look at how we can apply styles to our page at the page level. Thus far, you've seen how styles are applied, but you haven't seen any style sheets. Here's what one looks like:

```
<style type="text/css">
h1 { font-size: x-large; font-weight: bold }
h2 { font-size: large; font-weight: bold }
</style>
```

The `<style>` tag should be included within the `<head>` tag on your page. The `type` attribute indicates the MIME type of the style sheet. `text/css` is the only value you'll use. The body of the style sheet consists of a series of rules. All rules follow the same structure:

```
selector { property1: value1; property2: value2; .. }
```

Each rule consists of a selector followed by a list of properties and values associated with those properties. All the properties being set for a selector are enclosed in curly braces, as shown in the example. You can include any number of properties for each selector, and they must be separated from one another using semicolons. You can also include a semicolon following the last property/value pair in the rule, or not—it's up to you.

You should already be quite familiar with CSS properties and values because that's what you use in the `style` attribute of tags. Selectors are something new. I discuss them in detail in a bit. The ones I've used thus far have the same names as tags. If you use `h1` as a selector, the rule will apply to any `<h1>` tags on the page. By the same token, if you use `p` as your selector, it will apply to `<p>` tags.

Creating Sitewide Style Sheets

You can't capture the real efficiency of style sheets until you start creating sitewide style sheets. You can store all of your style information in a file and include it without resorting to any server trickery (which I discuss in Lesson 21, "Taking Advantage of the Server"). A CSS file contains the body of a `<style>` tag. To turn the style sheet from the previous section into a separate file, you could just save the following to a file called `style.css`:

```
h1 { font-size: x-large; font-weight: bold }
h2 { font-size: large; font-weight: bold }
```

In truth, the extension of the file is irrelevant, but the extension `.css` is the de facto standard for style sheets, so you should probably use it. After you've created the style sheet file, you can include it in your page using the `<link>` tag, like this:

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

The `type` attribute is the same as that of the `<style>` tag. The `href` tag is the same as that of the `<a>` tag. It can be a relative URL, an absolute URL, or even a fully qualified URL that points to a different server. As long as the browser can fetch the file, any URL will work. This means that you can just as easily use other people's style sheets as your own, but you probably shouldn't.

There's another attribute of the link tag, too: `media`. This enables you to specify different style sheets for different display mediums. For example, you can specify one for print, another for screen display, and others for things like aural browsers for use with screen readers. Not all browsers support the different media types, but if your style sheet is specific to a particular medium, you should include it. The options are `screen`, `print`, `projection`, `aural`, `braille`, `tty`, `tv`, `embossed`, `handheld`, and `all`.

You can also specify titles for your style sheets using the `title` attribute, as well as alternative style sheets by setting the `rel` attribute to `alternative style sheet`. Theoretically, this means that you could specify multiple style sheets for your page (with the one set to `rel="stylesheet"` as the preferred style sheet). The browser would then enable the user to select from among them based on the title you provide. Unfortunately, none of the major browsers support this behavior.

As it is, you can include links to multiple style sheets in your pages, and all the rules will be applied. This means that you can create one general style sheet for your entire site and then another specific to a page or to a section of the site, too.

As you can see, the capability to link to external style sheets provides you with a powerful means for managing the look and feel of your site. After you've set up a sitewide style sheet that defines the styles for your pages, changing things such as the headline font and background color for your pages all at once is trivial. Before CSS, making these kinds of changes required a lot of manual labor or a facility with tools that had search and replace functionality for multiple files. Now it requires quick edits to a single-linked style sheet.

Selectors

You've already seen one type of selector for CSS: element names. Any tag can serve as a CSS selector, and the rules associated with that selector will be applied to all instances of that tag on the page. You can add a rule to the `<b>` tag that sets the font weight to normal if you choose to do so, or italicize every paragraph on your page by applying a style to the `<p>` tag. Applying styles to the `<body>` tag using the `body` selector enables you to apply pagewide settings. However, you can apply styles on a more granular basis in a number of ways and to apply them across multiple types of elements using a single selector.

First, there's a way to apply styles to more than one selector at the same time. Suppose, for instance that you want all unordered lists, ordered lists, and paragraphs on a page displayed using blue text. Instead of writing individual rules for each of these elements, you can write a single rule that applies to all of them. Here's the syntax:

```
p, ol, ul { color: blue }
```

A comma-separated list indicates that the style rule should apply to all the tags listed. The preceding rule is just an easier way to write the following:

```
p { color: blue }
ol { color: blue }
ul { color: blue }
```

Contextual Selectors

Contextual selectors are also available. These are used to apply styles to elements only when they're nested within other specified elements. Take a look at this rule:

```
ol em { color: blue }
```

The fact that I left out the comma indicates that this rule applies only to `em` elements that are nested within ordered lists. Let's look at two slightly different rules:

```
p cite { font-style: italic; font-weight: normal }
li cite { font-style: normal; font-weight: bold }
```

In this case, <cite> tags that appear within <p> tags will be italicized. If a <cite> tag appears inside a list item, the contents will be rendered in boldface. Let's add in one more rule:

```
cite { color: green }
p cite { font-style: italic; font-weight: normal }
li cite { font-style: normal; font-weight: bold }
```


In this case, we have one rule that applies to all <cite> tags, and the two others that you've already seen. In this case, the contents of all <cite> tags will be green, and the appropriately nested <cite> tags will take on those styles, too. Here's one final example:

```
cite { color: green }
p cite { font-style: italic; font-weight: normal; color: red }
li cite { font-style: normal; font-weight: bold; color: blue }
```

In this case, the nested styles override the default style for the <cite> tag. The contents of <cite> tags that don't meet the criteria of the nested rules will appear in green. The nested rules will override the color specified in the less-specific rule, so for <cite> tags that are inside <p> tags, the contents will be red. Inside list items, the contents will be blue.

The ability to override property settings by using more specific selectors is what provides the ability to set styles with the precision of the `style` attribute from a style sheet.

Classes and IDs

Sometimes selecting by tag (even using contextual selectors) isn't specific enough for your needs, and you must create your own classifications for use with CSS. There are two attributes supported by all HTML tags: `class` and `id`. The `class` attribute is used to classify elements, and the `id` attribute is for assigning identifiers to specific elements.

To apply a selector to a class, use a leading `.` in the class name in your style sheet. So, if you have a tag like this

```
<div class="important">Some text.</div>
```

then you write the rule like this

```
.important { color: red; font-weight: bold; }
```

Any element with the class `important` will appear in bold red text. If you want to give this treatment to only important s, you can include the element name along with the class name in your rule.

```
div.important { color: red; font-weight: bold; }
p.important { color: blue; font-weight: bold; }
```

In this case, if a <p> tag is has the class important, the text inside will be blue. If a has the important class, its text will be red. You could also rewrite the preceding two rules as follows:

```
.important { font-weight: bold; }
div.important { color: red; }
p.important { color: blue; }
```

All members of the important class will be bold and important s will be red, whereas important paragraphs will be blue. If you assigned the important class to another tag, like <li>, the default color would be applied to it.

Whenever you want to specify styles for a single element, assign it an ID. As you'll learn later in the book, assigning IDs to elements is also very useful when using JavaScript because doing so lets you write scripts that reference individual items specifically. For now, however, let's look at how IDs are used with CSS. Generally, a page will have only one footer. To identify it, use the id attribute:

```
<div id="footer">
Copyright 2010, Example Industries.
</div>
```

You can then write CSS rules that apply to that element by referencing the ID. Here's an example:

```
#footer { font-size: small; }
```

As you can see, when you refer to IDs in your style sheets, you need to prepend a # on the front to distinguish them from class names and element names. Note that there's no additional facility for referring to IDs that are associated with particular elements. IDs are supposed to be unique, so there's no need for qualifying them further. Finally, there's nothing to say that you can't mix up all these selectors in one rule, like so:

```
h1, #headline, .heading, div.important { font-size: large; color: green; }
```

As you can see, I've included several types of selectors in one rule. This is perfectly legal if you want to set the same properties for a number of different selectors. Classes also work with contextual selectors:

```
ul li.important { color: red }
```

In this case, list items in the important class will be red if they occur in an unordered list. If they're in an ordered list, the rule will not be applied.

CAUTION

One common mistake is to include the `.` when assigning classes or the # when assigning IDs. The punctuation should only be used in the style sheet. In the attributes, leave them off. So `id="primary"` is correct, `id="#primary"` is not.

What Cascading Means

You may be wondering where the *cascading* in Cascading Style Sheets comes from. They are so named because styles cascade from parent elements to their children. To override a style that has been applied via cascading, you just need to set the same property using a more specific selector.

Here's an example style sheet that will illustrate how cascading works:

```
body { font-size: 200% }
div { font-size: 80% }
p { font-size: 80% }
span.smaller { font-size: 80%; font-weight: bold; }
#smallest { font-size: 80%; font-weight: normal; }
```

Figure 8.1 shows what the page looks like when that style sheet is applied to the following HTML:

Input ▼

```
<div>
  This text is in a div but not in a paragraph.

  <p>This test is in a paragraph.</p>

  <p><span class="smaller">This is in a span with the class "smaller"
inside a paragraph.</span></p>

  <p><span class="smaller"><span id="smallest">This text is in a
span with the ID "smallest".</span></span></p>
</div>
```

Output ▶

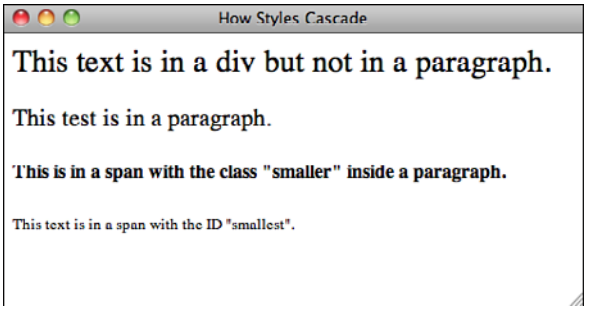


FIGURE 8.1
How cascading styles work.

When percentage units are used in style sheets, the percentage is applied to the value that's inherited as the styles cascade down. To start, all the text on the page is set to a font size of 200% using the selector for the `<body>` tag. Then I use a variety of selectors to make the text progressively smaller as the styles cascade down through the style sheet. With CSS, the styles that are applied to a given element are calculated from all the selectors that match that style in the style sheet.

It's also possible to override styles. This style sheet sets the font weight for spans with the class `smaller` to `bold`. The element with the ID `smallest` has its font weight set to normal. In Figure 8.1, you'll see that the last line is not bold. It inherits the font weight from the `span.smaller` selector, but the `#smallest` selector overrides it.

There are a number of other selectors that enable you to apply styles very specifically without requiring you to add your own classes or IDs to elements. Lesson 13, "Advanced CSS Techniques," discusses those.

Units of Measure

One of the most confusing aspects of CSS is the units of measure it provides. Four types of units can be specified in CSS: length units, percentage units, color units, and URLs.

There are two kinds of length units: absolute and relative. *Absolute* units theoretically correspond to a unit of measure in the real world, such as an inch, a centimeter, or a point. *Relative* units are based on some more arbitrary unit of measure. Table 8.1 contains a full list of length units.

TABLE 8.1 Length Units in CSS

Unit	Measurement
`em`	Relative; height of the element's font
`ex`	Relative; height of x character in the element's font

TABLE 8.1 Continued

Unit	Measurement
px	Relative; pixels
in	Absolute; inches
cm	Absolute; centimeters
mm	Absolute; millimeters
pt	Absolute; points
pc	Absolute; picas

The absolute measurements seem great, except that an inch isn't really an inch when it comes to measuring things on a screen. Given the variety of browser sizes and resolutions supported, the browser doesn't really know how to figure out what an inch is. For example, you might have a laptop with a 14.1" display running at 1024 by 768. I might have a 20" CRT running at that same resolution. If the browser thinks that one inch is 96 pixels, a headline set to `1in` may appear as less than an inch on your monitor or more than an inch on mine. Dealing with relative units is safer.

In this lesson, I use one length unit: `px`. It's my favorite for sizing most things. However, other relative units can also be useful. For example, if you want paragraphs on your page to appear as double spaced, you can specify them like this:

```
p { line-height: 2em; }
```

Percentage units are also extremely common. They're written as you'd expect: `200%` (with no spaces). The thing to remember with percentages is that they're always relative to something. If you set a font size to 200%, it will be double the size of the font it inherited through CSS, or 200% of the browser's default font size if a font size that has been applied to that element. If you set a `<div>`'s width to 50%, it will be half as wide as the enclosing element (or the browser window, if there's no enclosing element). When you use percentages, always keep in mind what you're talking about a percent of.

Using Percentage Units

When you use percentages as units, bear in mind that the percentage applies not to the size of the page, but rather to the size of the box that encloses the box to which the style applies. For example, if you have a `<div>` with its width set to 50% inside a `<div>` with its width set to `500px`, the inner `<div>` will be 250 pixels wide. On the other hand, if the outer `<div>` were also set to 50%, it would be half as wide as the browser window, and the inner `<div>` would be 25% of the width of the browser window.

Colors can be specified in a variety of ways, all of which are discussed in Lesson 9, "Adding Images, Color, and Backgrounds." Some colors can be specified by name, and that's the method I've used so far in this lesson. Most of the time, when you use URLs, they're used in the `<a>` tag or `<img>` tag. In CSS, they're usually included to specify the location of a background image or a bullet image for a list. Generally, URLs are specified like this:

```
url(http://www.example.com/)
```

The Box Model

When working with CSS, it helps to think of every element on a page as being contained within a box. This is true of inline elements like `<a>` or block-level elements like `<p>`. Each of these boxes is contained within three larger boxes, and the entire set of four is referred to as the CSS box model. Figure 8.2 shows a diagram of the box model.

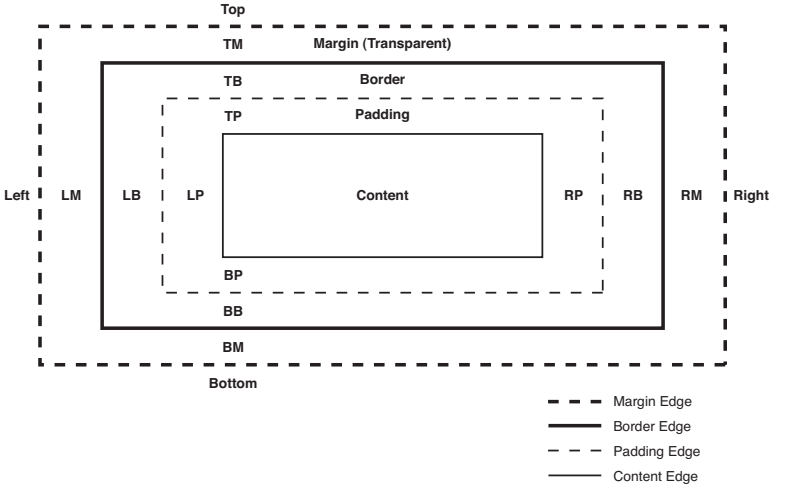


FIGURE 8.2
The CSS box model.

The innermost box contains the content of the element. Surrounding that is the padding, then the border, and finally, the outermost layer (the margin). In addition to properties that you can use to change how the content is displayed, CSS provides properties that can be used to change the padding, border, and margins around each box. In this section, you learn how to modify all the layers in the box model. If you get confused about how the layers are ordered, just refer back to Figure 8.2.

Borders

Before I talk about padding or margins, I want to talk about borders. CSS provides several properties for adding borders around elements and changing how they are displayed. Using CSS, you can apply a border to any box.

The `border-style` property specifies the type of border that will be displayed. Valid options for the `border-style` are `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, and `inherit`. Most of the styles alter the border appearance, but `none` and `inherit` are special. Setting the `border-style` to none disables borders, and `inherit` uses the `border-style` inherited from a less-specific selector.

The `border-width` property specifies how wide the border around a box should be. Borders are usually specified in pixels, but any CSS unit of measurement can be used. To create a 1-pixel, dashed border around all the anchors on a page, you use the following CSS:

```
a { border-width: 1px; border-style: solid; }
```

The final border style, `border-color`, is used to set the color for a border. To set the border color for links to red, you use the following style declaration:

```
a { border-color: red; }
```

You can also set border properties for an element using what's called a *shortcut property*. Instead of using the three separate border properties, you can apply them all simultaneously as long as you put the values in the right order, using the `border` property. It's used as follows:

```
selector { border: style width color; }
```

So, to add a three pixel dashed red border to the links on a page, you use the following style decoration:

```
a { border: dashed 3px red; }
```

You can use different values for each side of a box when you're using any of the box properties. There are two ways to do so. The first is to add directions to the property names, as follows:

```
a {
  border-width-left: 3px;
  border-style-left: dotted;
  border-color-left: green;
}
```

The directions are `top`, `bottom`, `left`, and `right`. Alternatively, you can set the values for each side. If you specify four values, they will be applied to the top, right, bottom, and left, in that order. If you specify two values, they will be applied to the top and bottom and left and right. To set different border widths for all four sides of a box, you use the following style:

```
p.box { border-width: 1px 2px 3px 4px; }
```

That's equivalent to the following:

```
p.box {
  border-width-top: 1px;
  border-width-right: 2px;
  border-width-bottom: 3px;
  border-width-left: 4px;
}
```

To apply different values for the `border` shortcut property to different sides of a box, it's necessary to use the directional property names. You can't supply multiple values for the components of the shortcut property. However, the directional properties are treated as being more specific than the general border property, so you can use styles like this:

Input ▼

```
p {
  border: solid  2px red ;
  border-bottom: dashed 4px green;
}
```

The results are in Figure 8.3.

Output ▶

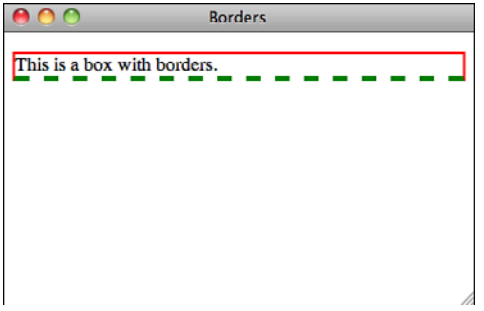


FIGURE 8.3
Border styles.

Margins and Padding

In the box model, there are two ways to control whitespace around a box. Padding is the whitespace inside the border, and the margin is the whitespace outside the border, separating the box from surrounding elements. Let's look at an example that illustrates how padding and margins work. The web page that follows has one `<div>` nested within another. The outer `<div>` has a solid black border; the inner `<div>` has a dotted black border. The page appears in Figure 8.4.

Input ▼

```
<html>
<head>
  <title>Nested Elements</title>
  <style type="text/css">
    .outer { border: 2px solid black; }
    .inner { border: 2px dotted black;
         padding: 0;
         margin: 0; }
  </style>
</head>
<body>
<div class="outer">
Outer.
<div class="inner">
Friends, Romans, countrymen, lend me your ears;<br />
I come to bury Caesar, not to praise him.<br />
The evil that men do lives after them;<br />
The good is oft interred with their bones;<br />
So let it be with Caesar. The noble Brutus<br />
</div>
</div>

</body>
</html>
```

Output ▶

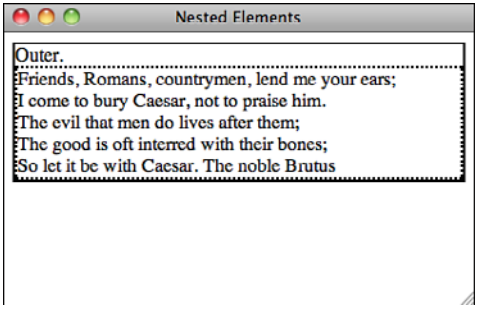


FIGURE 8.4
Nested * *s with no margins or padding.

As you can see, the text in the inner `<div>` is jammed right up against the border, and the inner border and outer border are flush against each other. That's because I've set both the padding and margin of the inner `<div>` to `0`. (When you're setting a property to `0` there's no need to specify a unit.) The results in Figure 8.5 show what happens if I change the style sheet to this:

Input ▼

```
.outer { border: 2px solid black; }
.inner { border: 2px dotted black;
     padding: 15px;
     margin: 15px; }
```

Output ▶

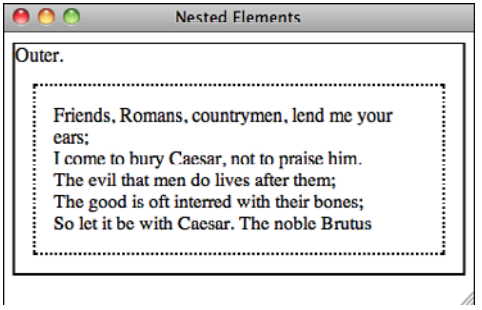


FIGURE 8.5
The inner * * has 15 pixels of padding and margin here.

As you can see, I've created some space between the border of the inner `<div>` and the text inside the inner `<div>` using `padding`, and some space between the border of the inner `<div>` and the border of the outer `<div>` using `margin`. Now let's look at what happens when I add some margin and padding to the outer `<div>`, too. I'm also going to give both the inner and outer `<div>`s background colors so that you can see how colors are assigned to whitespace. (I discuss backgrounds and background colors in the next lesson.) The results are in Figure 8.6. Here's the new style sheet:

Input ▼

```
.outer { border: 2px solid black;
     background-color: gray;
     padding: 15px;
     margin: 40px; }
.inner { border: 2px dotted black;
     background-color: white;
     padding: 15px;
     margin: 15px; }
```

Output ▶

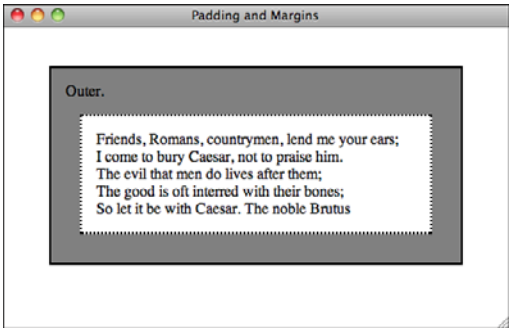


FIGURE 8.6
Both the inner and outer have margin and padding.

I gave the outer a large 40-pixel margin so that you could see how it moves the borders away from the edges of the browser window. Note also that there's now space between the text in the outer and the border. You can also see that the padding of the outer and the margin of the inner are combined to provide 30 pixels of whitespace between the border of the outer and the border of the inner . Finally, it's important to understand the behavior of the background color. The background color is applied to the padding but not to the margin. So, the 15-pixel margin outside the inner takes on the background color of the outer , and the margin of the outer takes on the background color of the page.

Collapsing Margins

In the CSS box model, horizontal margins are never collapsed. (If you put two items with horizontal margins next to each other, both margins will appear on the page.) Vertical margins, on the other hand, are collapsed. Only the larger of the two vertical margins is used when two elements with margins are next to each other. For example, if a with a 40-pixel bottom margin is above a with a 20-pixel top margin, the margin between the two will be 40 pixels, not 60 pixels.

You already know that to center text within a box, the `text-align` property is used. The question now is this: How do you center a box on the page? In addition to passing units of measure or a percentage to the margin property, you can also set the margin to `auto`. In theory, this means set this margin to the same value as the opposite margin. However, if you set both the left and right margins to `auto`, your element will be centered. To do so, you can use the `margin-left` and `margin-right` properties or provide multiple values for the `margin` property. So, to center a horizontally, the following style sheet is used. (The newly centered is in Figure 8.7.)

Input ▼

```
.inner { border: 2px dotted black;
     background-color: white;
     padding: 15px;
     width: 50%;
     margin-left: auto;
     margin-right: auto;
     }
```

Output ▶

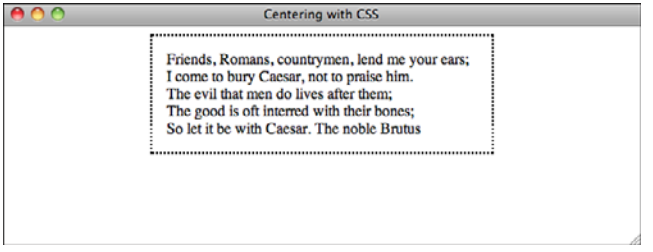


FIGURE 8.7
A centered `<div>`.

CAUTION

Browsers care about your Document Type Definition (DTD) settings. For example, if you don't indicate in your document that you're using HTML 4.01, XHTML 1.0, or HTML5, Internet Explorer will not honor things such as `margin: auto`. If the DTD is left out, IE assumes that you're using an old version of HTML that doesn't support features like that. You should always use a DTD that correctly indicates which version of HTML is used on the page.

TIP

If you want elements to overlap each other, you can apply negative margins to them rather than positive margins.

I used the `width` property in that style sheet to shrink the `<div>` so that it could be centered. I explain how to resize elements using CSS later in the lesson.

Another thing to remember is that the `<body>` of the page is a box, too. Here's a style sheet that includes new values for the `border`, `margin`, and `padding` properties of the `<body>` tag. It also includes some changes to the outer `<div>` to illustrate how the changes to the `<body>` tag work. You can see the updated page in Figure 8.8.

Input ▼

```
.outer { border: 2px solid black;
     background-color: gray;
     padding: 15px; }
.inner { border: 2px dotted black;
     background-color: white;
     padding: 15px;
     margin: 15px; }
body { margin: 20px;
    border: 3px solid blue;
    padding: 20px;
    background-color: yellow;
```

Output ▶

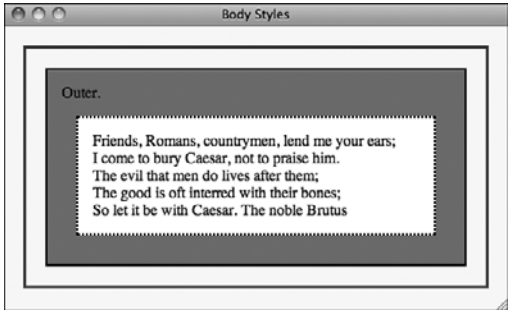


FIGURE 8.8
Treating the body of a document as a box.

In this example, you can see that you can adjust the margin, padding, and border of a document's body. However, unlike other boxes, the background color is applied to the margin as well as the padding.

Controlling Size and Element Display

The one box in the box model I haven't discussed is the content box. For starters, there are two types of content boxes: block and inline. In previous lessons, I've discussed block-level elements versus inline elements—this distinction is important in CSS. Block elements are, by default, as wide as the container you place them within, and you can modify their height and width using CSS. Block elements are also preceded and followed by line breaks. Inline elements are only as wide as they need to be to display their contents, as well as the margins, borders, and padding that you apply to them.

Each element is, by default, either a block element or an inline element, but CSS provides the `display` property to allow you to change this behavior. The block property supports three values: `block`, `inline`, and `none`. For example, if you want the elements in a list to appear inline rather than each appearing on its own line, as shown in Figure 8.9, you use the following style:

```
ul.inline li { display: inline }
```

FIGURE 8.9
Inline list items.

Setting the `display` property to `none` removes the selected elements from the page entirely. Hiding elements with this property is useful if you want to use JavaScript to dynamically hide and show items on the page. Using JavaScript to modify page styles is discussed starting in Lesson 14, "Introducing JavaScript."

There are two properties for controlling the size of a block: `width` and `height`. They enable you to set the size of the box using any of the units of measurement mentioned previously. If you use a percentage for the height or width, that percentage is applied to the size of the containing element.

To make the header of your page 100 pixels high and half the width of the browser, you could use the following rule:

```
#header { width: 50%; height: 100px; }
```

The following paragraph will appear to be very narrow, but the box in which it resides will be as wide as the browser window unless you specify a width.

```
<p>one.<br />two.<br />three.<br /></p>

It's possible to set maximum and minimum heights and widths for elements to
account for differences in the size of users' browser windows. The properties
that enable you to do so are max-width, min-width, max-height, and min-height.
Let's say you've created a page design that only looks right if it's at least
600 pixels wide. You could use the following style:#container { min-width: 600px
}
```

The element with the ID `container` will expand to fit the size of the browser window as long as it's at least 600 pixels wide. If the browser is smaller than 600 pixels wide, the contents of the element will scroll off the screen. Likewise, you may want to constrain the maximum size of an element so that lines of text do not become so long that they're difficult to read. To do so, use the following style:

```
#container { max-width: 800px }
```

You can also use both styles together to keep the size of your page within a certain range, regardless of the size of the user's browser window:

```
#container { min-width: 600px; max-width: 800px; }
```

Normally elements in HTML are sized to fit the content that is placed within them. However, if you constrain the size of an element with a size or a maximum size and then place content inside the element that won't fit, the browser has to decide what to do with it. By default, when content won't fit inside its box, the browser just displays the overflow as best it can. As you can see from Figure 8.10, the results are not always pretty.

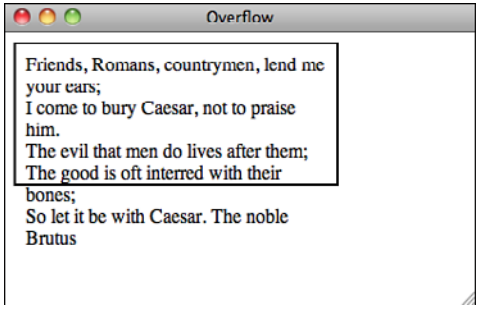


FIGURE 8.10
Content that is too large for its container.

The border shows the dimensions of the box specified in the style sheet. Because there's too much text to fit inside the box, it runs over the border and down the page. Using the CSS `overflow` property, you can tell the browser what to do when these situations arise. The values are `visible` (this is the default), `hidden`, `scroll`, `auto`, and `inherit`. You can see how the different overflow settings look in Figure 8.11.

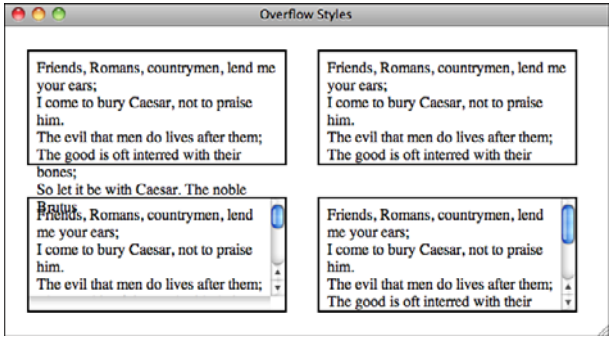


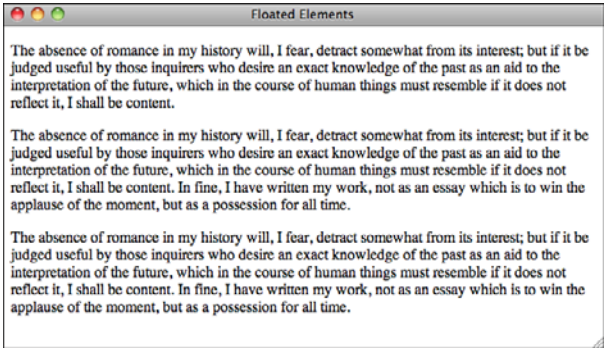
FIGURE 8.11
Different ways of dealing with overflow.

When overflow is `hidden`, the content that does not fit in the box is not displayed at all. Both `scroll` and `auto` add scrollbars to enable users to view the entire contents of the box. When the setting is `scroll`, the scrollbars are always displayed, whereas when the setting is `auto`, the scrollbars display only if needed. When overflow is visible, content that overflows the box is not taken into account when laying out other items on the page and the overflow content bleeds onto other content on the page. When you are sizing elements on the page manually, you should always account for potential overflow so that it doesn't break the layout of your page.

Float

Normally, block-level elements flow down the page from top to bottom. If you want to alter the normal flow of the page, you can use absolute positioning, which I discuss in a bit, or you can use the `float` property. The `float` property is used to indicate that an element should be placed as far as possible to the left or right on the page, and that any other content should wrap around it. This is best illustrated with an example. First, take a look at the page in Figure 8.12.

FIGURE 8.12
A page with no floating elements.



As you can see, the three boxes run straight down the page. I've added a border to the first box, but that's it. Here's the source code to the page, with the addition of a few additional properties that demonstrate how `float` works:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
  <title>Floated Elements</title>
```

```
  <style type="text/css" media="screen">
    .right {
      border: 3px solid black;
      padding: 10px;
      margin: 10px;
      float: right;
      width: 33%; }

    .bottom { clear: both; }
  </style>
</head>
<body>
<p class="right">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content.
</p>
<p class="main">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content. In fine, I have written
my work, not as an essay which is to win the applause of the moment,
but as a possession for all time.
</p>
<p class="bottom">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content. In fine, I have written
my work, not as an essay which is to win the applause of the moment,
but as a possession for all time.
</p>
</body>
</html>
```

As you can see from the style sheet, I've set the `float` property for elements with the class "right" to `right`. I've also added some padding, a margin, and a border to that class for aesthetic purposes and set the width for that class to 33% so that it isn't as wide as the browser window. I've also put the second paragraph on the page in the class `bottom`, and I've added the `clear: both` property to it. Figure 8.13 shows the results.

Output ▶

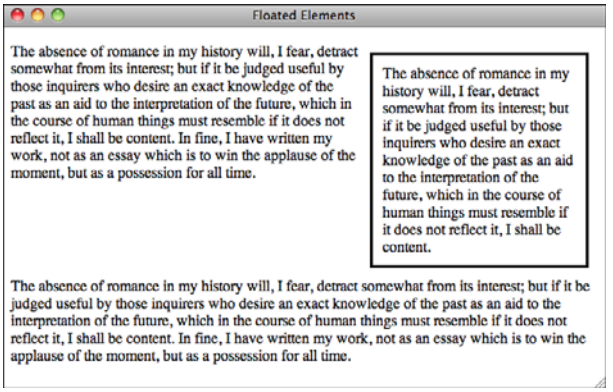


FIGURE 8.13
A page with a <*div*> floated to the right.

The `<div>` is moved over to the right side of the page, and the first paragraph appears next to it. The `float: right` property indicates that the rest of the page's content should flow around it. The bottom paragraph does not flow around the `<div>` because I've applied the `clear: both` property to it, which cancels any float that has been set. The options for `float` are easy to remember: `left`, `right`, and `none`. The options for clear are `none`, `left`, `right`, and `both`.

Using the `clear` property, you have the option of clearing either the left or right float without canceling both at the same time. This proves useful if you have a long column on the right and a short one on the left and you want to maintain the float on the right even though you're canceling it on the left (or vice versa).

Now let's look at how floated elements work together. Figure 8.14 shows what happens when you have two right-floating elements together, and Figure 8.15 shows the effect with a left-floating element and a right-floating element.

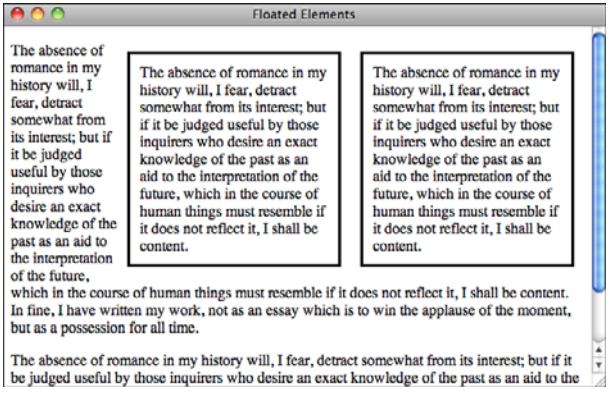


FIGURE 8.14
Two right-floating elements together.

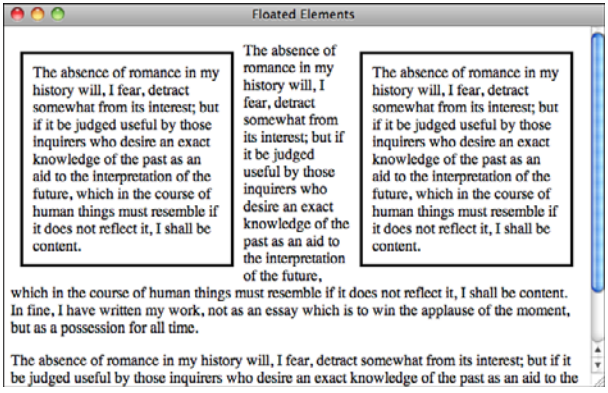


FIGURE 8.15
A left-floating and a right-floating element together.

As you can see, when you put two floating elements together, they appear next to each other. If you want the second one to appear below the first, you need to use the `clear` property as well as the `float` property in the rule, as shown in this style sheet:

Input ▼

```
.right {
  border: 3px solid black;
  padding: 10px;
  margin: 10px;
  float: right;
  width: 33%; }

#second { clear: right; }

.bottom { clear: both; }
```

The additional `<div>` I've added has been given the ID second, so that it inherits all the styles of the class `right` and also the style rule associated with the ID `second`. The result is in Figure 8.16.

Output ▶

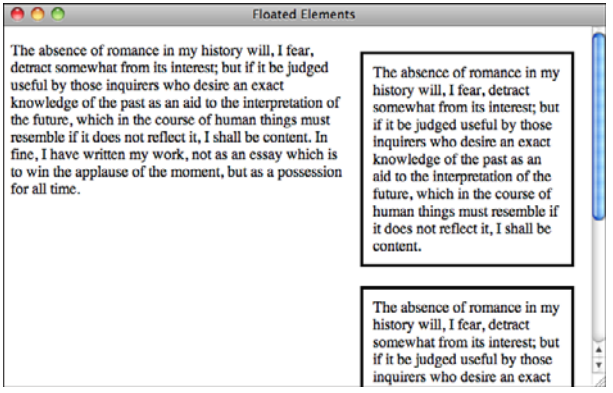


FIGURE 8.16
Two floating elements that are aligned vertically.

CSS Positioning

If using `float` to control how elements are laid out doesn't provide the measure of control you're looking for, you can use the CSS positioning attributes. To position elements yourself, you first have to choose a positioning scheme with the `position` property. There are four positioning schemes, three of which you'll actually use. The four are `static`, `relative`, `absolute`, and `fixed`.

The `static` scheme is the default. Elements flow down the page from left to right and top to bottom, unless you use the `float` property to change things up. The `relative` scheme positions the element relative to the element that precedes it. You can alter the page flow to a certain degree, but the elements will still be interdependent. The `absolute` and `fixed` schemes enable you to position elements in any location you want on the page. The `fixed` scheme is not well supported, so if you want to control where items appear yourself you should use `absolute`.

After you've picked a positioning scheme, you can set the position for elements. There are four positioning properties: `top`, `left`, `bottom`, and `right`. The values for these properties are specified as the distance of the named side from the side of the enclosing block. Here's an example:

```
.thing {
    position: relative;
    left: 50px;
    top: 50px;
}
```

In this case, elements in the `thing` class will be shifted 50 pixels down and 50 pixels to the left from the elements that precede them in the page layout. If I were to change `position` to `absolute`, the element would appear 50 pixels from the top-left corner of the page's body.

Generally, when you're positioning elements, you pick a corner and specify where the element should be located. In other words, there's never a need to set more than two of the four positioning properties. If you set more than two, you could run into problems with how the browser renders the page because you're specifying not only where the element should be located but also the size of the element. It's much safer to use the sizing properties to size your elements and then specify the position of one corner of your element if you want to indicate where it should go on the page.

Relative Positioning

Let's look at a page that uses relative positioning. This page illustrates both how relative positioning works and some of the problems with it. A screenshot of the page listed in the following code appears in Figure 8.17.

Input ▼

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Example</title>
  <style type="text/css">
.offset {
  border: 3px solid blue;
  padding: 10px;
  margin: 10px;
  background-color: gray;
  position: relative;
  top: -46px;
  left: 50px;
  width: 33%; }
  </style>
</head>
<body>
<p>
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content.
</p>
<p class="offset">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content. In fine, I have written
my work, not as an essay which is to win the applause of the moment,
```

```
but as a possession for all time.
</p>
</body>
</html>
```

Output ▶

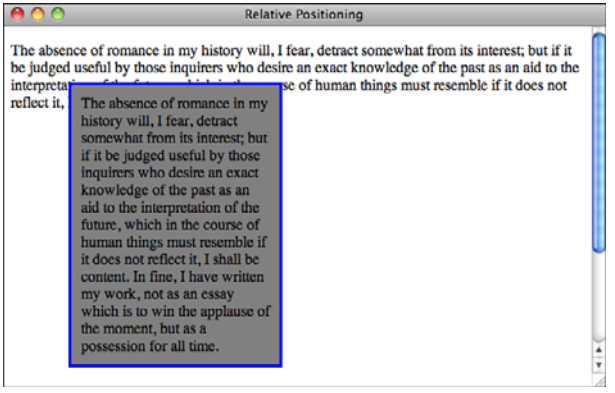


FIGURE 8.17
A page that uses relative positioning for an element.

You can spot the problem right away on this page—the relatively positioned element overlaps the paragraph above it. I used a negative value for the `top` property to move the element up 50 pixels, and also moved it to the left 50 pixels. The hole where the content would normally be positioned if I were using static positioning remains exactly the same as it would had I not moved the element, thus creating whitespace before the third paragraph. However, due to the positioning, the paragraph has been moved up so that it overlaps the one above it.

By default, the backgrounds of elements on an HTML page are transparent. I added a background color to the relatively positioned box to more clearly illustrate how my page works. If I remove the `background-color` property from class `two`, the page looks like the one in Figure 8.18.

In this example, transparency is probably *not* the effect I'm looking for. However, taking advantage of this transparency can be a useful effect when you create text blocks that partially overlap images or other nontext boxes.

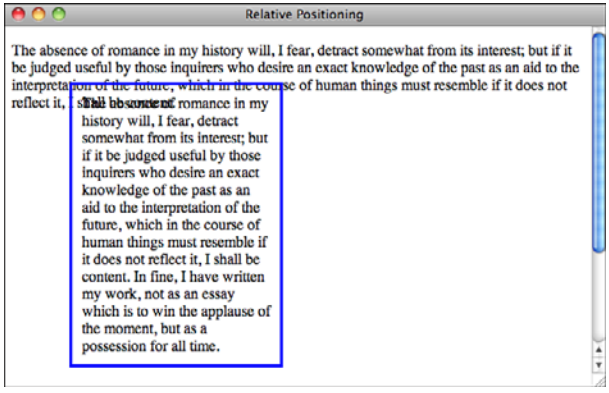


FIGURE 8.18
Transparency of overlapping elements.

Absolute Positioning

Now let's look at absolute positioning. The source code for the page shown in Figure 8.19 contains four absolutely positioned elements.

Input ▼

```
<!DOCTYPE html>
<html>
<head>
  <title>Absolute Positioning</title>
  <style type="text/css">
#topleft {
  position: absolute;
  top: 0px;
  left: 0px;
}

#topright {
  position: absolute;
  top: 0px;
  right: 0px;
}

#bottomleft {
  position: absolute;
  bottom: 0px;
  left: 0px;
}
```

```
#bottomright {
  position: absolute;
  bottom: 0px;
  right: 0px;
}

.box {
  border: 3px solid red;
  background-color: #blue;
  padding: 10px;
  margin: 10px;
}
  </style>
</head>
<body>
<div class="box" id="topleft">
Top left corner.
</div>

<div class="box" id="topright">
Top right corner.
</div>

<div class="box" id="bottomleft">
Bottom left corner.
</div>

<div class="box" id="bottomright">
Bottom right corner.
</div>

<p>
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content.
</p>
<p>
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content. In fine, I have written
my work, not as an essay which is to win the applause of the moment,
but as a possession for all time.
</p>
</body>
</html>
```

Output ▶

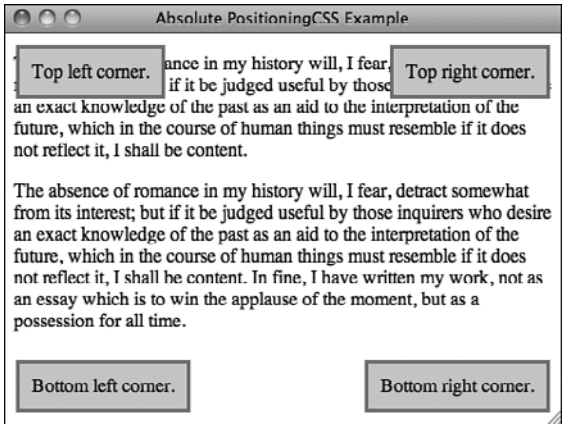


FIGURE 8.19
A page that uses absolute positioning.

Aside from the fact that there are now four absolutely positioned `<div>`s on the page exactly where I indicated they should be placed, a couple of other things should stand out here. The first item to notice is that when you absolutely position elements, there's no placeholder for them in the normal flow of the page. My four `<div>`s are defined at the top, and yet the first paragraph of the text starts at the beginning of the page body. Unlike relative positioning, absolute positioning completely removes an element from the regular page layout. The second interesting fact is that absolutely positioned elements overlap the existing content without any regard for it. If I want the text in the paragraphs to flow around the positioned elements, I have to use `float` rather than `position`.

Not only can I use any unit of measurement when positioning elements, I can also use negative numbers if I choose. You already saw how I applied a negative value to the top of the relatively positioned element to move it up some; I can do the same thing with these absolutely positioned elements. The result of changing the rule for the `topleft` class in the earlier example to

Input ▼

```
#topleft {
  position: absolute;
  top: -30px;
  left: -30px;
}
```

is that it actually pulls the element partially off of the page, where it is inaccessible even using the scrollbars, as shown in Figure 8.20.

Output ▶

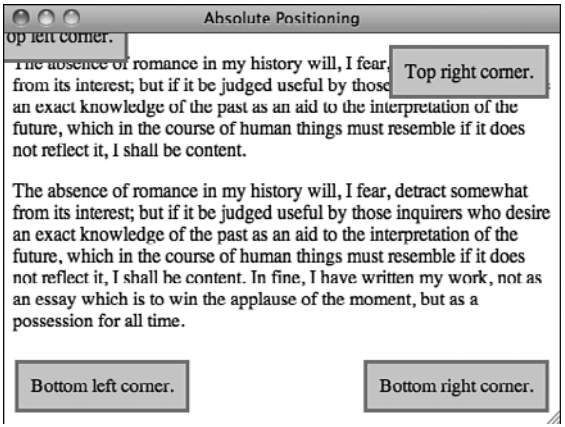


FIGURE 8.20
Use of negative absolute positioning.

Controlling Stacking

CSS provides a way of taking control over how overlapping elements are presented. The `z-index` property defines stacking order for a page. By default, elements that appear in the same layer of a document are stacked in source order. In other words, an element that appears after another in the HTML source for the page will generally be stacked above it.

By assigning `z-index` values to elements, however, you can put elements in specific stacking layers. If all elements appear in stacking layer 0 by default, any element in stacking layer 1 (`z-index: 1`) will appear above all elements in layer 0. The catch here is that `z-index` can be applied only to elements that are placed using absolute or relative positioning. Elements that are placed using static positioning always appear below relatively or absolutely positioned elements. The stacking layers below 0 are considered beneath the body element, and so they don't show up at all.

TIP

If you want to have an element positioned as though it were part of the static positioning scheme but you want to control its stacking layer, assign it the relative positioning scheme and don't specify a position. It will appear on the page normally but you will be able to apply a `z-index` to it.

Let's look at another page. This one contains two paragraphs, both part of the same (default) stacking layer. As you can see in Figure 8.21, the second overlaps the first.

Input ▼

```
<!DOCTYPE html>
<html>
<head>
  <title>Stacking Example</title>
  <style type="text/css">
.one {
  position: relative;
  width: 50%;
  padding: 15px;
  background-color: yellow;
}

.two {
  position: absolute;
  top: 15%;
  left: 15%;
  padding: 15px;
  width: 50%;
  background-color: navy;
  color: white;
}
  </style>
</head>
<body>
<p class="one">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content.
</p>
<p class="two">
The absence of romance in my history will, I fear, detract somewhat
from its interest; but if it be judged useful by those inquirers who
desire an exact knowledge of the past as an aid to the interpretation
of the future, which in the course of human things must resemble if
it does not reflect it, I shall be content. In fine, I have written
my work, not as an essay which is to win the applause of the moment,
but as a possession for all time.
</p>

</body>
</html>
```

Output ▶

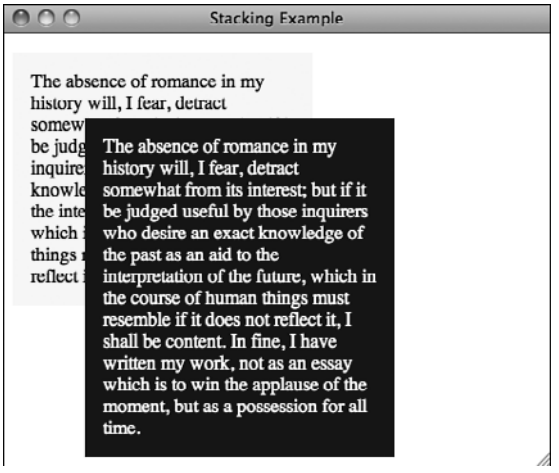


FIGURE 8.21
Two normally stacked elements.

So, how do I cause the first element to overlap the second? Because I've assigned the first element the relative positioning scheme (even though I haven't positioned it), I can assign it a `z-index` of `1` (or higher) to move it into a stacking layer above the second paragraph. The new style sheet for the page, which appears in Figure 8.22, is as follows:

Input ▼

```
.one {
  position: relative;
  z-index: 1;
  width: 50%;
  padding: 15px;
  background-color: #ffc;
}

.two {
  position: absolute;
  top: 15%;
  left: 15%;
  padding: 15px;
  width: 50%;
  background-color: #060;
  color: #fff;
}
```

Output ▶

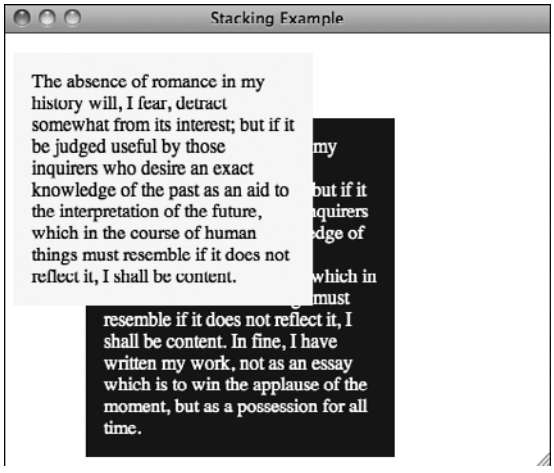


FIGURE 8.22
A page that uses *z-index* to control positioning.

Using a combination of absolute and relative positioning, you can create very complex pages with many stacked layers.

The <body> Tag

I've already mentioned that you can adjust the margin, padding, and border of a page by applying styles to the <body> tag. More important, any styles that you want to apply on a pagewide basis can be assigned to the page's body. You already know about setting the background color for the page by using `style="background-color: black"` in your <body> tag. That's really just the beginning. If you want the default font for all the text on your page to appear in the Georgia font, you can use the following style:

```
body { font-family: Georgia; }
```

That's a lot easier than changing the `font-family` property for every tag that contains text on your page. A common <body> tag you often see looks something like this:

```
<body bgcolor="#000000" text="#ffffff" alink="blue" vlink="yellow" alink="
  purple">
```

You can modify the background and text colors like this:

```
body { color: white;
    background-color: black; }
```

I explain how to alter the link colors shortly. One of the main advantages of taking this approach, aside from the fact that it's how the standard says you should do things, is that then you can put the style into a linked style sheet and set the background color for your whole site on one page.

Many layouts require that elements be flush with the edge of the browser. In these cases, you need to set the margin to `0` for your `<body>` tag. Some browsers enabled you to do this with proprietary attributes of the `<body>` tag, but they're not reliable. To turn off margins, just use this rule:

```
body { margin: 0px; }
```

Links

You already know how to adjust the colors of elements on a page, but links are a bit different. They're more complicated than other types of elements because they can exist in multiple states: an unvisited link, a visited link, an active link, and a link that the user currently has the pointer over. As you can see, there's one more state here than has been traditionally reflected in the `<body>` tag. Using CSS, you can change the color of a link when the user mouses over it (referred to as the *hover* state) as opposed to when he's currently clicking it (the *active* state).

Another advantage of CSS is that you can change the color schemes for links on the same page, rather than being forced to use one scheme throughout. Finally, you can turn off link underlining if you want. For example, here's a style sheet that turns off link underlining for navigation links, renders them in boldface, and keeps the same color for visited and unvisited links:

```
a:link  { color: blue; }
a:active { color: red; }
a:visited { color: purple; }
a:hover  { color: red; }
a.nav   { font-weight: bold;
      text-decoration: none; }
a.nav:hover, a.nav: active { background-color: yellow;
               color: red; }
a.nav:link, a.nav:visited { color: green; }
```

From the style sheet, you can see that for all `<a>` tags in the class `nav`, the `text-decoration` property is set to `none`, which turns off underlining, and `font-weight` is set to `bold`. For `<a>` tags on the rest of the page, the underlining remains on, but I've set it up so that when the mouse is over the links, they turn red. For navigation links, when the mouse is over the links, the background of the element turns yellow and the text turns red.

CAUTION

You can use pretty much any property you like with the pseudo-selectors for links, and browsers that support them will dynamically reflow the page to accommodate the change. However, changes that affect the size of the element (such as boldfacing the text dynamically or increasing the font size) can be jarring to users, so use them cautiously.

Summary

In the preceding lessons, I've given you a taste of how to use CSS. You didn't get the full flavor because I used them only within the context of the `style` attribute of tags. In this lesson, I discussed how you can create style sheets either as part of a page or as a stand-alone file that can be included by any page. I also moved beyond properties that discuss text formatting to explain how to use CSS to lay out an entire page.

By understanding how browsers render pages and how you can affect that process using CSS, you can achieve the effects you want without writing loads of markup that's difficult to understand and maintain.

You'll continue to be introduced to new CSS properties in the lessons that follow. In Lesson 9, I explain how to use CSS to change colors on the page, and provide all the details on using CSS to define the backgrounds of pages and specific elements. Lesson 13 takes a deeper look at CSS selectors and explains how to create entire page layouts using CSS.

Workshop

In this lesson, you learned about Cascading Style Sheets, the wonderful supplement to HTML that makes formatting your pages less painful. Throughout the rest of this book, I use CSS where appropriate, so please review this workshop material before continuing.

Q&A

Q My CSS isn't working like I'd expect. What should I do?

A CSS probably doesn't seem that clear in the first place, and things can only get messier when you actually start applying styles to your pages. You should be sure to test your pages in every browser you can find, and don't be afraid to experiment. Just because something seems like it should work doesn't mean it will. The W3C also provides a CSS Validator (http://jigsaw.w3.org/css-validator/) that you can use to make sure that your CSS syntax is correct. You should probably use it all the time, but even if you don't, it can still help out if you get stuck.

Q Are there naming rules for classes and IDs?

A Yes, there are. A name must start with a letter, and can contain only letters, numbers, or dashes (-). Some browsers may not enforce these rules, but to be safe, you should adhere to them.

Q What are the relevant CSS standards?

A There are three CSS recommendations from the W3C: CSS1, CSS2, and CSS3. Most modern browsers support a large part of CSS1 and CSS2, as well as parts of CSS3. You can find out more at http://www.w3.org/Style/CSS/. If you're curious about how well your browser supports CSS or the effect that properties have in real browsers, you can check out the CSS test suites at http://www.w3.org/Style/CSS/Test/. CSS2 and CSS3 include a number of additional selectors. They are discussed in Lesson 13.

Quiz

1. Why can't absolute units be used reliably in CSS?
2. True or false: Including style sheets on your page requires features provided by a web server.
3. How do the absolute and relative positioning schemes differ?
4. Is the margin or padding of an element inside the border?
5. How do you lay out your page so that elements positioned statically appear above elements that are positioned absolutely?

Quiz Answers

1. Absolute units have problems in CSS because there's no way to know exactly what sort of display medium the user has. An inch on one monitor might be completely different than an inch on another.
2. The answer is false; you can use the `<link>` tag to load external style sheets without involving the web server in any way.
3. The relative positioning scheme places elements relative to the previous element on the page, whereas the absolute positioning scheme places the element exactly where you tell it to on the page.
4. The padding of an element is inside the border of an element, and the margin is outside.

5. This is a trick question. You cannot put statically positioned elements above absolutely positioned elements. However, if you change the statically positioned elements so that they use the relative positioning scheme, you can alter their stacking layer using the `z-index` property.

Exercises

1. If you've already created some web pages, go back and try to figure out how you could apply CSS to them.
2. Examine the style sheets used by some websites that you admire. Take a look at how they use classes and IDs in their markup.
3. Create a web page that includes a sidebar on the left, with text wrapped around it. Create a navigation menu at the bottom that is positioned below the sidebar.

This page intentionally left blank

LESSON 9

Adding Images, Color, and Backgrounds

Few things can do more to make a Web page more interesting than a strategically placed image or an attractive color scheme. Effective use of images and color is one of the key things that separates professionally designed sites from those designed by novices. The process of selecting images, resizing them and saving them in the proper format, and integrating them into a page can be intimidating, but this lesson explains how it's done.

This lesson covers the following topics:

- The kinds of images you can use in web pages
- How to include images on your web page, either alone or alongside text
- How to use images as clickable links
- How to set up and assign links to regions of images using client-side imagemaps
- How to provide alternatives for browsers that can't view images
- How to change the font and background colors on your web page
- How to use images for tiled page backgrounds
- How and when to use images on your web pages
- A few tips on image etiquette

Images on the Web

Images displayed on the Web should be converted to one of the formats supported by most browsers: GIF, JPEG, or PNG. GIF and JPEG are the popular standards, and every graphical browser supports them. PNG is a newer image format that was created in response to some patent issues with the GIF format and is widely used, too. Many other image formats are supported by some browsers and not others. You should avoid them.

Let's assume that you already have an image you want to put on your web page. How do you get it into GIF or JPEG format so it can be viewed on your page? Most image editing programs, such as Adobe Photoshop (http://www.adobe.com/), iPhoto (http://apple.com/), Picasa (http://picasa.google.com/) and GIMP (http://gimp.org/), convert images to most of the popular formats. You might have to look under the option for Save As or Export to find the conversion option. There are also freeware and shareware programs for most platforms that do nothing but convert between image formats. Many shareware and demo versions of image-editing programs are available at http://www.download.com/. Look under Image Editing Software in the Digital Photo Software section. There are also online photo editors available at `http://pixlr.com` and `http://picnik.com`.

TIP

If you're a Windows user, you can download IrfanView, which enables you to view images and convert them to various formats, at http://www.infanview.com/. It also provides a number of other image-manipulation features that are useful for working with images for the Web. Best of all, it's free for noncommercial use.

Remember how your HTML files need an `.html` or `.htm` extension to work properly? Image files have extensions, too. For PNG files, the extension is `.png`. For GIF files, the extension is `.gif`. For JPEG files, the extensions are `.jpg` and `.jpeg`.

CAUTION

Some image editors try to save files with extensions in all caps (`.GIF` or `.JPEG`). Although they're the correct extensions, image names are case-sensitive, so `.GIF` isn't the same extension as `.gif`. The case of the extension might not be important when you're testing on your local system, but it can be when you move your files to the server. So, use lowercase if you can.

Image Formats

As I just mentioned, three image formats are supported by every major web browser: GIF, JPEG, and PNG. JPEG and GIF are the old standbys, each useful for different purposes. PNG was designed as a replacement for the GIF format, which was necessary after Unisys invoked its patent rights on the GIF format. (The patent has since expired.) To design web pages, you must understand and apply both image formats and decide which is appropriate to use in each case.

GIF

Graphics Interchange Format, also known as GIF or CompuServe GIF, was once the most widely used image format. It was developed by CompuServe to fill the need for a cross-platform image format.

NOTE

GIF is pronounced *jiff*, like the peanut butter, not with a hard *G* as in *gift*. Really—the early documentation of GIF tools says so.

The GIF format is actually two similar image formats: GIF87, the original format, and GIF89a, which has enhancements for transparency, interlacing, and multiframe GIF images that you can use for simple animations.

The GIF format is okay for logos, icons, line art, and other simple images. It doesn't work as well for highly detailed images because it's limited to only 256 colors. For example, photographs in GIF format tend to look grainy and blotchy. The problem is that with the limited color palette, it's hard to create smooth color transitions.

JPEG

JPEG, which stands for *Joint Photographic Experts Group* (the group that developed it), is the other popular format for images on the Web. JPEG (pronounced *jay-peg*) is actually a compression type that other file formats can use. The file format for which it's known is also commonly called JPEG.

JPEG was designed for the storage of photographic images. Unlike GIF images, JPEG images can include any number of colors. The style of compression that JPEG uses (the compression algorithm) works especially well for photographs, so files compressed using the JPEG algorithm are considerably smaller than those compressed using GIF. JPEG uses a *lossy* compression algorithm, which means that some of the data used in the image

is discarded to make the file smaller. Lossy compression works extremely well for photographic data but makes JPEG unsuitable for images that contain elements with sharp edges, such as logos, line art, and type. JPEG files are supported by all major web browsers.

If you're working with photos to display on the web, you should save them in the JPEG format.

PNG

PNG, pronounced "ping," was originally designed as a replacement for GIFs. It stands for *Portable Network Graphics*. Only the oldest browsers don't support PNG natively. Current browsers all support PNG, and it has some important advantages over GIF (and to a lesser extent over JPEG). Like GIF, it is a nonlossy image format. No information about the image is lost when it is compressed.

It has better support for transparency than GIF and supports palette-based images (like GIF) and true-color and grayscale images (like JPEG). In other words, you don't have to worry about color usage with PNG, although limiting color usage can result in smaller files.

More and more sites use the PNG format for images, but due mainly to inertia, GIF and JPEG are still the most common formats. The good news is that nearly all graphics tools these days support PNG. If you're creating new images that aren't photographs, PNG is the format to use.

Inline Images in HTML: The `<img>` Tag

After you have an image ready to go, you can include it on your web page. Inline images are placed in HTML documents using the `<img>` tag. This tag, like the `<hr>` and `<br>` tags, has no closing tag in HTML. For XHTML and HTML5, you must add an extra space and forward slash to the end of the tag to indicate that it has no closing tag.

The `<img>` tag has many attributes that enable you to control how the image is presented on the page. Some attributes have been deprecated in favor of *Cascading Style Sheets (CSS)*.

NOTE

To use the `<img>` tag in an XHTML-compliant fashion, you need to close it, like this:

```
<img />
```

The most important attribute of the <img> tag is src, which is the URL of the image you want to include. Paths to images are specified in the same way as the paths in the href attribute of links. So, to point to a GIF file named image.gif in the same directory as the HTML document, you can use the following XHTML tag:

```
<img src="image.gif" />
```

For an image file one directory up from the current directory, use this XHTML tag:

```
<img src="../image.gif" />
```

And so on, using the same rules as for page names in the href part of the <a> tag.

You can also point to images on remote servers from the src attribute of an <img> tag, just as you can from the href attribute of a link. If you want to include the image example.gif from www.example.com on your web page, you could use the following tag:

```
<img src="http://www.example.com/example.gif" />
```

CAUTION

Just because you can use images stored on other servers for your own web pages doesn't mean that you should. A lot of legal, ethical, and technical issues are involved with using images on other sites. I discuss them later in this lesson.

Adding Alternative Text to Images

Images can turn a simple text-only web page into a glorious visual feast. But what happens if someone is reading your web page using a text-only browser? What if she has image loading turned off so that all your carefully crafted graphics appear as generic icons? All of a sudden, that visual feast doesn't look quite as glorious.

There's a simple solution to this problem. By using the alt attribute of the <img> tag, you can substitute something meaningful in place of the image on browsers that cannot display it.

In text-only browsers, such as Lynx, graphics that are specified using the <img> tag in the original file usually are displayed as the word IMAGE with square brackets around it, like this: [IMAGE]. If the image itself is a link to something else, that link is preserved.

The alt attribute in the <img> tag provides a more meaningful text alternative to the blank [IMAGE] for your visitors who are using text-only web browsers or who have graphics turned off on their browsers. The alt attribute contains a string with the text you want to substitute for the graphic:

```
<img src="myimage.gif" alt="[a picture of a cat]" />
```

Most browsers interpret the string you include in the `alt` attribute as a literal string. That is, if you include any HTML tags in that string, they'll be printed as-is rather than being parsed and displayed as HTML code. Therefore, you can't use whole blocks of HTML code as a replacement for an image—just a few words or phrases.

I bring up image alternatives now for good reason. The `alt` attribute is required in XHTML 1.0 but is optional in HTML5 . If there's no appropriate alternative text for an image, you can simply leave it empty, like this: `alt=""`.

▼ Task: Exercise 9.1: Adding Images to a Page

Here's the web page for a local haunted house that's open every year at Halloween. Using all the excellent advice I've given you in the preceding eight lessons, you should be able to create a page like this one fairly easily. Here's the HTML code for this HTML file, and Figure 9.1 shows how it looks so far:

Input ▼

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to the Halloween House of Terror</title>
  </head>
  <body>
    <h1>Welcome to The Halloween House of Terror</h1>
    <p>
      Voted the most frightening haunted house three years in a
      row, the <strong>Halloween House of Terror</strong>
      provides the ultimate in Halloween thrills. Over
      <strong>20 rooms of thrills and excitement</strong> to
      make your blood run cold and your hair stand on end!
    </p>
    <p>
      The Halloween House of Terror is open from <em>October 20
        to November 1st</em>, with a gala celebration on
        Halloween night. Our hours are:
    </p>
    <ul>
      <li>Mon-Fri 5PM-midnight</li>
      <li>Sat &amp;amp; Sun 5PM-3AM</li>
      <li><strong>Halloween Night (31-Oct)</strong>: 3PM-???</li>
    </ul>
    <p>
      The Halloween House of Terror is located at:<br />
      The Old Waterfall Shopping Center<br />
      1020 Mirabella Ave<br />
```

```
      Springfield, CA 94532
    </p>
  </body>
</html>
```

Output ▶

FIGURE 9.1
The Halloween House home page.

So far, so good. Now you can add an image to the page. Suppose that you happen to have an image of a haunted house lying around on your hard drive; it would look excellent at the top of this web page. The image, called `house.jpg`, is in JPEG format. It's located in the same directory as the `halloween.html` page, so adding it to the page will be easy.

Now, suppose that you want to place this image above the page heading. To do so, add an `<img>` tag to the file, just before the heading:

```
<div><img src="haunted_house.png" alt="House of Terror" /></div>
<h1>Welcome to The Halloween House of Terror</h1>
```

Images, like links, don't define their own text elements, so the `<img>` tag has to go inside a paragraph or heading element.

When you reload the `halloween.html` page, your browser should include the haunted house image on the page, as shown in Figure 9.2.

If the image doesn't load and your browser displays a funny-looking icon in its place, make sure that you entered the filename properly in the HTML file. Image filenames are case-sensitive, so all the uppercase and lowercase letters have to be correct.

If the case isn't the problem, double-check the image file to make sure that it is indeed a GIF or JPEG image and that it has the proper file extension.

FIGURE 9.2
The Halloween House home page with the haunted house.

If one image is good, two would be really good, right? Try adding another `<img>` tag next to the first one, as follows, and see what happens:

Input

```
<div><img src="haunted_house.png" alt="House of Terror" />
<img src="haunted_house.png" alt="House of Terror" /></div><h1>Welcome to The
Halloween House of Terror</h1>
```

Figure 9.3 shows how the page looks in a browser. The two images are adjacent to each other, as you would expect.

Output

FIGURE 9.3
Multiple images.

And that's all there is to adding images!

Images and Text

In the preceding exercise, you put an inline image on a page with text below it. You also can include an image inside a line of text. This is what the phrase "inline image" actually means—it's *in* a *line* of text.

To include images inside a line of text, just add the `<img>` tag inside an element tag (`<h1>`, `<p>`, `<address>`, and so on), as in the following line:

```
<h2><img src="house.jpg" alt="House of Terror" />The Halloween House of
Terror!!</h2>
```


Figure 9.4 shows the difference you can make by putting the image inline with the heading. (I've also shortened the heading and changed it to `<h2>` so that it all fits on one line.)

FIGURE 9.4
The Halloween House page with an image inside the heading.

The image doesn't have to be large, and it doesn't have to be at the beginning of the text. You can include an image anywhere in a block of text, as in the following:

Input ▼

```
<blockquote>
      Love, from whom the world <img src="world.gif" /> begun,<br />
      Hath the secret of the sun. <img src="sun.gif" /><br />
      Love can tell, and love alone, Whence the million stars
      <img src="star.gif" /> were strewn<br />
      Why each atom <img src="atom.gif" /> knows its own.<br />
      —Robert Bridges
    </blockquote>
```

Figure 9.5 shows how this block looks.

Output ▶

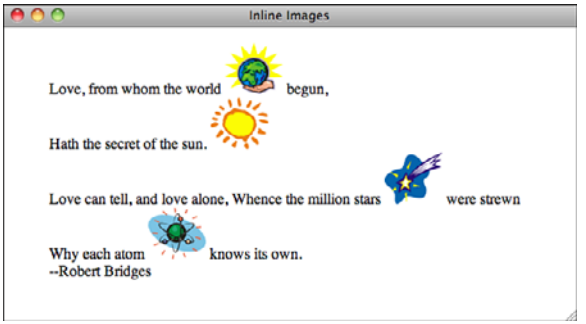


FIGURE 9.5
Images can go anywhere in text.

Text and Image Alignment

In these examples, the bottom of the image and the bottom of the text match up. The `<img>` tag also includes the `align` attribute, which enables you to align the top or bottom of the image with the surrounding text or other images in the line.

NOTE

> The `align` attribute for the `<img>` tag was deprecated in HTML 4.01 in favor of using CSS and has been dropped from HTML5 completely. However, you may find yourself using it if you're using a content management system or some other application to publish Web content. In many cases it's easier to modify your markup than it is to alter the style sheets for the site. Alternatively, you can adjust the `style` attribute with your `img` tags.

Standard HTML 2.0 defines three basic values for `align`:

`align="top"`	Aligns the top of the image with the topmost part of the line (which may be the top of the text or the top of another image). This is the same as `vertical-align: top` in CSS.
`align="middle"`	Aligns the center of the image with the middle of the line (usually the baseline of the line of text, not the actual middle of the line). This is the same as `vertical-align: middle` in CSS.
`align="bottom"`	Aligns the bottom of the image with the bottom of the line of text. This is the same as `vertical-align: bottom` in CSS.

There are also two other values: `left` and `right`. These values are discussed in the next section, "Wrapping Text Next to Images."

Figure 9.6 shows the Robert Bridges poem from the previous section with the world image unaligned, the sun image aligned to the top of the line, the star image aligned to the middle, and the atom aligned to the bottom of the text.

Input ▼

```
<blockquote>
      Love, from whom the world
      <img src="world.gif" /> begun,<br />
      Hath the secret of the sun.
      <img src="sun.gif" align="top" /><br />
      Love can tell, and love alone, Whence the million stars
      <img src="star.gif" align="middle" /> were strewn<br />
      Why each atom <img src="atom.gif" align="bottom" />
      knows its own.<br />
      —Robert Bridges
</blockquote>
```

Output ▶

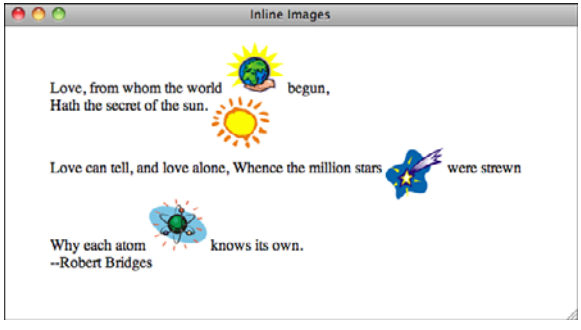


FIGURE 9.6
Images unaligned, aligned top, aligned middle, and aligned bottom.

In addition to the preceding values, several other nonstandard values for `align` provide greater control over precisely where the image will be aligned within the line. The following values are unevenly supported by browsers and have been superseded by CSS equivalents, too.

`align="texttop"`	Aligns the top of the image with the top of the tallest text in the line (whereas `align="top"` aligns the image with the topmost item in the line). Use `vertical-align: text-top` in CSS instead.
`align="absmiddle"`	Aligns the middle of the image with the middle of the largest item in the line. (`align="middle"` usually aligns the middle of the image with the baseline of the text, not its actual middle.) No CSS equivalent.

`align="baseline"`	Aligns the bottom of the image with the baseline of the text. `align="baseline"` is the same as `align="bottom"`, but `align="baseline"` is a more descriptive name. This is the default vertical alignment and can be written in CSS as `vertical-align: baseline`.
`align="absbottom"`	Aligns the bottom of the image with the lowest item in the line (which may be below the baseline of the text). Replaced in CSS by `vertical-align: text-bottom`.

The following code shows these alignment options at work:

Input ▼

```
<h2>Middle of Text and Line aligned, arrow varies:</h2>
    <p>
      <img src="line.gif" />
      Align: Top <img src="uparrow.gif" align="top" />
      Align: Text Top <img src="uparrow.gif" align="texttop" />
    </p>
    <h2>Top of Text and Line aligned, arrow varies:</h2>
    <p>
      <img src="line.gif" />
      Align: Absolute Middle <img src="forward.gif" align="absmiddle" />
      Align: Middle <img src="forward.gif" align="middle" />
    </p>
    <h2>Top of Text and Line aligned, arrow varies:</h2>
    <p>
      <img src="line.gif" />
      Align: Baseline / Bottom <img src="down.gif" align="baseline" />
      Align: Absolute Bottom <img src="down.gif" align="absbottom" />
    </p>
```

Figure 9.7 shows examples of all the options as they appear in a browser. In each case, the line on the left side and the text are aligned with each other, and the position of the arrow varies.

Output ▶

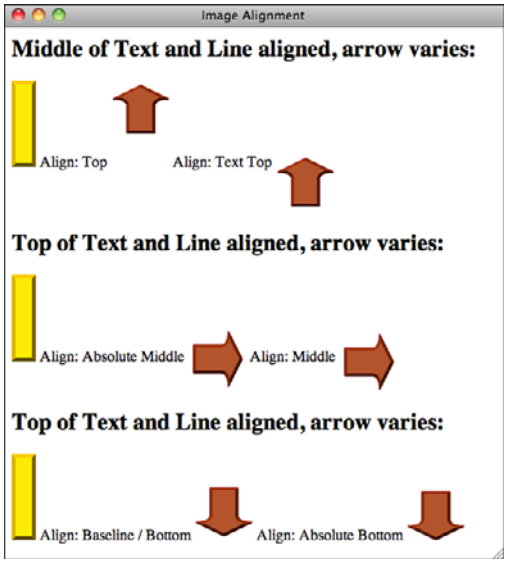


FIGURE 9.7
Image alignment options.

Wrapping Text Next to Images

Including an image inside a line works fine if you have only one line of text. To control the flow of text around an image, you need to use CSS or the `align` attribute of the `img` tag. Images are just like any other element as far as the float property goes, so you can use the `float` and `clear` CSS properties to control text flow around them, as discussed in the previous lesson. HTML5 Vdoes not support the `align` attribute, but all the browsers that are currently in use do, and many people still use it.

`align="left"` and `align="right"`

`align="left"` aligns an image with the left margin, and `align="right"` aligns an image with the right margin. However, these attributes also cause any text that follows the image to be displayed in the space to the right or left of that image, depending on the margin alignment:

Input ▼

```
<p><img src="tulips.gif" align="left" /></p>
<h1>Mystery Tulip Murderer Strikes</h1>
<p>
  Someone, or something, is killing the tulips of New South
  Haverford, Virginia. Residents of this small town are
  shocked and dismayed by the senseless vandalism that has
  struck their tiny town.
</p>
<p>
```

```
    New South Haverford is known for its extravagant displays
    of tulips in the springtime, and a good portion of its
    tourist trade relies on the people who come from as far as
    New Hampshire to see what has been estimated as up to two
    hundred thousand tulips that bloom in April and May.
  </p>
  <p>
    Or at least the tourists had been flocking to New South
    Haverford until last week, when over the course of three
    days the flower of each and every tulip in the town was
    neatly clipped off while the town slept.
  </p>
  <p>
    "It started at the south end of town," said Augustin Frouf,
    a retired ladder-maker who has personally planted over five
    hundred pink lily-flowered tulips. "They hit the houses up
    on Elm Street, and moved down into town from there. After
    the second night, we tried keeping guard. We tried bright
    lights, dogs, everything. There was always something that
    pulled us away, and by the time we got back, they were all
    gone."
  </p>
```

Figure 9.8 shows an image with some text aligned next to it.

You can put any HTML text (paragraphs, lists, headings, other images) after an aligned image, and the text will be wrapped into the space between the image and the margin. Or you can have images on both margins and put the text between them. The browser fills in the space with text to the bottom of the image and then continues filling in the text beneath the image.

Output ▶

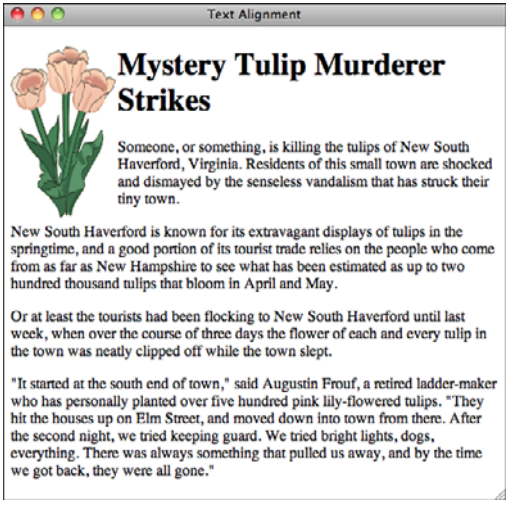


FIGURE 9.8
Text and images aligned.

Stopping Text Wrapping

What if you want to stop filling in the space and start the next line underneath the image? A normal line break won't do it; it just breaks the line to the current margin alongside the image. A new paragraph also continues wrapping the text alongside the image. To stop wrapping text next to an image, use the `clear` CSS property. This enables you to stop the line wrapping where appropriate.

As mentioned in Lesson 8, "Using CSS to Style a Site," the `clear` property can have one of three values:

`left`	Break to an empty left margin, for left-aligned images
`right`	Break to an empty right margin, for right-aligned images
`all`	Break to a line clear to both margins

For example, the following code snippet shows a picture of a tulip with some text wrapped next to it. Adding a `style` attribute to the first paragraph with `clear` set to `left` breaks the text wrapping after the heading and restarts the text after the image:

Input ▼

```
<p><img src="tulips.gif" align="left" /></p>
    <h1>Mystery Tulip Murderer Strikes</h1>
    <p style="clear: left">
      Someone, or something, is killing the tulips of New South
      Haverford, Virginia. Residents of this small town are
      shocked and dismayed by the senseless vandalism that has
      struck their tiny town.
    </p>
    <p>
      New South Haverford is known for its extravagant displays
      of tulips in the springtime, and a good portion of its
      tourist trade relies on the people who come from as far as
      New Hampshire to see what has been estimated as up to two
      hundred thousand tulips that bloom in April and May.
    </p>
    <p>
      Or at least the tourists had been flocking to New South
      Haverford until last week, when over the course of three
      days the flower of each and every tulip in the town was
      neatly clipped off while the town slept.
    </p>
    <p>
      "It started at the south end of town," said Augustin Frouf,
      a retired ladder-maker who has personally planted over five
      hundred pink lily-flowered tulips. "They hit the houses up
      on Elm Street, and moved down into town from there. After
```

```
    the second night, we tried keeping guard. We tried bright
    lights, dogs, everything. There was always something that
    pulled us away, and by the time we got back, they were all
    gone."
</p>
```

Figure 9.9 shows the result in a browser.

Output ▶

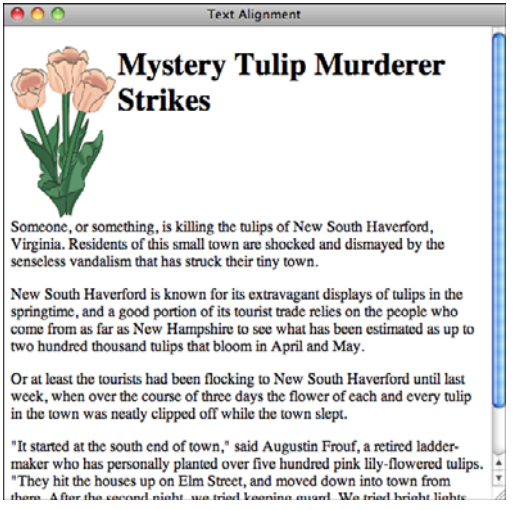


FIGURE 9.9
Line break to a clear margin.

Adjusting the Space Around Images

With the capability to wrap text around an image, you also might want to add some space between the image and the text that surrounds it. In the previous lesson, you learned how to manage the whitespace around elements using CSS padding and margins. When you're creating new pages, you should use CSS. Before CSS, the `vspace` and `hspace` attributes enabled you to make these adjustments, and you may still see people use them. Both take values in pixels; `vspace` controls the space above and below the image, and `hspace` controls the space to the left and the right. Note that the amount of space you specify is added on both sides of the image. For example, if you use `hspace="10"`, 10 pixels of space will be added on both the left and right sides of the image. It's the equivalent of the CSS `padding-left: 10px` and `padding-right: 10px`.

NOTE

The `vspace` and `hspace` attributes for the `<img>` tag are not included in HTML5. You should use the `margin` and `padding` CSS properties instead.

The following HTML code, displayed in Figure 9.10, illustrates two examples. The upper example shows default horizontal and vertical spacing around the image, and the lower example shows the effect produced by the `hspace` and `vspace` attributes. Both images use the `align="left"` attribute so that the text wraps along the left side of the image. However, in the bottom example, the text aligns with the extra space above the top of the image (added with the `vspace` attribute).

Input ▼

```
<p><img src="eggplant.gif" align="left" /></p>
<p>
  This is an eggplant. We intend to stay a good ways away
  from it, because we really don't like eggplant very much.
</p>
<p style="clear: left">
  <img src="eggplant.gif" vspace="50" hspace="50" align="left" />
</p>
<p>
  This is an eggplant. We intend to stay a good ways away
  from it, because we really don't like eggplant very much.
</p>
```

Output ▶

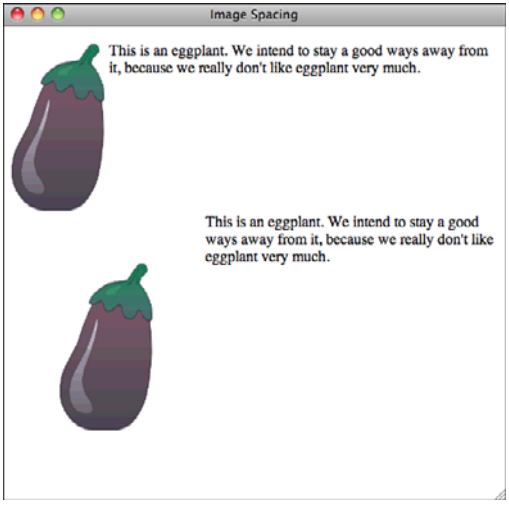


FIGURE 9.10
The upper example doesn't have image spacing, and the lower example does.

Here's the same example, written using CSS instead:

```
<img src="eggplant.gif" alt="Eggplant" style="float: left" />
<p>This is an eggplant. We intend to stay a good ways away from
it, because we really don't like eggplant very much.</p>
<hr style="clear: left" />
```

```
<img src="eggplant.gif" alt="Eggplant" style="float: left; padding: 50px"
/>
<p>This is an eggplant. We intend to stay a good ways away from
it, because we really don't like  eggplant very much. </p>
```

Images and Links

Can an image serve as a link? Sure it can! If you include an <img> tag inside a link tag (<a>), that image serves as a link itself:

```
<a href="index.html"><img src="uparrow.gif" alt="Up" /></a>
```

If you include both an image and text in the link tag, they become links to the same page:

```
<a href="index.html"><img src="uparrow.gif" alt="Up" />Up to Index</a>
```

TIP

One thing to look out for when you're placing images within links, with or without text, is whitespace between the tag and the <img> tag or between the text and the image. Some browsers turn the whitespace into a link, and you get an odd "tail" on your images. To avoid this unsightly problem, don't leave spaces or line feeds between your <img> tags and tags.

Some browsers put a border around images that are used as links by default, but not all of them. Figure 9.11 shows an example of this. The butterfly image is a not wrapped in a link, so it doesn't have a border around it. The up arrow, which takes the visitor back to the home page, has a border around it because it's a link.

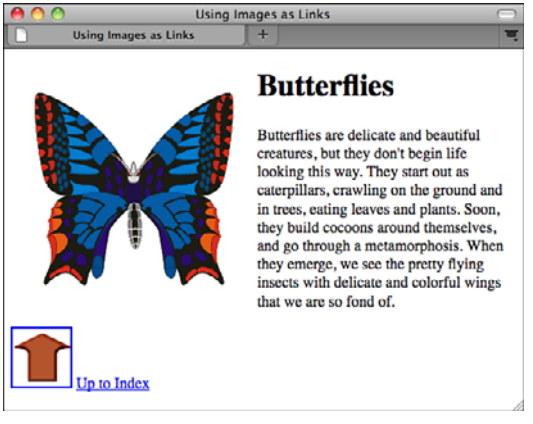


FIGURE 9.11
Images used as links sometimes have a border around them.

You can change the width of the border around the image by using the `border` attribute to `<img>` or the `border` CSS property. Support for the `border` attribute has been dropped from HTML5, so in this lesson, I use the CSS version. This attribute takes a number, which is the width of the border in pixels. `border="0"` hides the border entirely. You were introduced to the various properties that make up the `border` CSS property in the previous lesson. Disabling borders is ideal for image links that actually look like clickable buttons, as shown in Figure 9.12.

Because the default behavior differs between browsers when it comes to borders around linked images, you should always specify the border properties explicitly so that all users experience your site in the same way.

NOTE

Including borders around images that are links has really fallen out of favor with most web designers. Not turning them off can make your design look very dated.

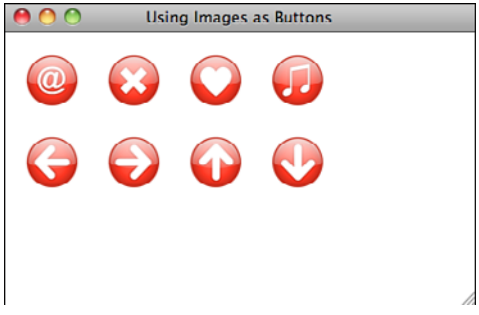


FIGURE 9.12
Images that look like buttons.

Task: Exercise 9.2: Using Navigation Icons

Now you can create a simple page that uses images as links. When you have a set of related web pages, it's usually helpful to create a consistent navigation scheme that is used on all the pages.

This example shows you how to create a set of icons that are used to navigate through a linear set of pages. You have three icons in GIF format: one for forward, one for back, and a third to enable the visitors to jump to the top-level contents page.

First, you write the HTML structure to support the icons. Here, the page itself isn't important, so you can just include a shell page:

Input

```
<!DOCTYPE html>
<html>
  <head>
    <title>Motorcycle Maintenance: Removing Spark Plugs</title>
  </head>
  <body>
    <h1>Removing Spark Plugs</h1>
    <p>(include some info about spark plugs here)</p>
  </body>
</html>
```

Figure 9.13 shows how the page looks at the beginning.

Output

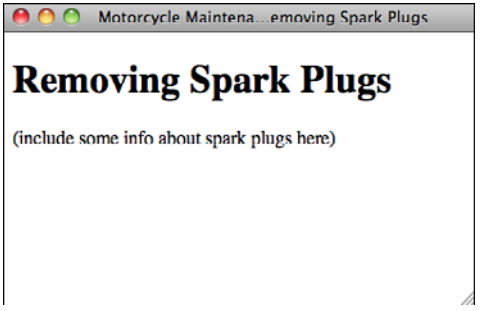


FIGURE 9.13
The basic page, with no icons.

At the bottom of the page, add your images using `<img>` tags:

Input

```
<div>
      <img src="Up.png" alt="Up" />
      <img src="Left.png" alt="Left" />
      <img src="Right.png" alt="Right" />
    </div>
```

Now add the anchors to the images to activate them:

Input

```
    <div>
      <a href="index.html"><img src="Up.png" alt="Up" /></a>
      <a href="ready.html"><img src="Left.png" alt="Left" /></a>
      <a href="replacing.html"><img src="Right.png" alt="Right" /></a>
    </div>
```

Figure 9.14 shows the result of this addition.

When you click the icons now, the browser jumps to the linked page just as it would have if you had used text links.

Output ▶

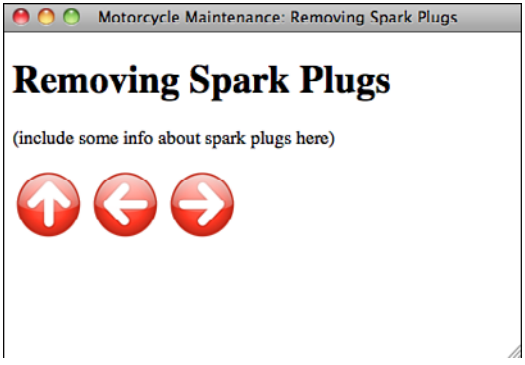


FIGURE 9.14
The basic page with navigation links.

Speaking of text, are the icons usable enough as they are? How about adding some text describing exactly what's on the other side of each link? You can add this text inside or outside the anchor, depending on whether you want the text to be a hot spot for the link, too. Here, include it outside the link so that only the icon serves as the hot spot. You also can align the bottoms of the text and the icons using the `align` attribute of the `<img>` tag. Finally, because the extra text causes the icons to move onto two lines, arrange each one on its own line instead:

Input ▼

```
<div>
  <a href="index.html"><img src="Up.png" alt="Up" /></a>
  Up to index
</div>
<div>
  <a href="ready.html"><img src="Left.png" alt="Left" /></a>
  On to "Gapping the New Plugs"
</div>
<div>
  <a href="replacing.html"><img src="Right.png" alt="Right" /></a>
  Back to "When You Should Replace your Spark Plugs"
</div>
```

See Figure 9.15 for the final menu.

Output ▶

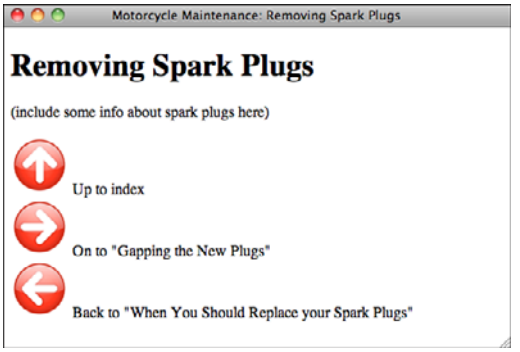


FIGURE 9.15
The basic page with iconic links and text.

▲

Other Neat Tricks with Images

Now that you've learned about inline images, images as links, and how to wrap text around images, you know what *most* people do with images on web pages. But you can play with a few newer tricks, too.

Image Dimensions and Scaling

Two attributes of the `<img>` tag, `height` and `width`, specify the height and width of the image in pixels. Both were part of the HTML 3.2 specification, but they're deprecated in favor of CSS now.

If the values for `width` and `height` are different from the actual width and height of the image, your browser will resize the image to fit those dimensions. The downside of this technique is that the image-scaling algorithms built into browsers are not always the best, generally images resized with graphics programs look better than images resized in the browser. If you use the browser to change the size of an image, be prepared for it to look pretty bad, especially if the aspect ratio is not preserved. (In other words, you take a 100-by-100 pixel image and expand it into a 200-by-400 pixel image.)

CAUTION

Don't perform *reverse scaling*—creating a large image and then using `width` and `height` to scale it down. Smaller file sizes are better because they take less time to load. If you're going to display a small image, make it smaller to begin with.

Here's an example <img> tag that uses the height and width attributes:

```
<img src="my_image.png" height="100" width="200" />
```

To specify the height and width using CSS, use the height and width properties:

```
<img src="my_image.png" style="height: 100px; width: 100px" />
```

If you leave out either the height or the width, the browser will calculate that value based on the value you provide for the other aspect. If you have an image that's 100 pixels by 100 pixels and you specify a height (using the attribute or CSS) of 200 pixels, the browser will automatically scale the width to 200 pixels, as well, preserving the original aspect ratio of the image. To change the aspect ratio of an image, you must specify both the height and width.

More About Image Borders

You learned about the border attribute of the <img> tag as part of the section on links, where setting border to a number or to 0 specified the width of the image border (or hid it entirely).

By default, images that aren't inside links don't have borders. However, you can use the border attribute to include a border around any image, as follows:

```
<p>
      <img src="eggplant.gif" align="left" style="border: 5px solid black">
      This is an eggplant. We intend to stay a good ways away from it,
      because we really don't like eggplant very much.
    </p>
```

Figure 9.16 shows an image with a border around it.

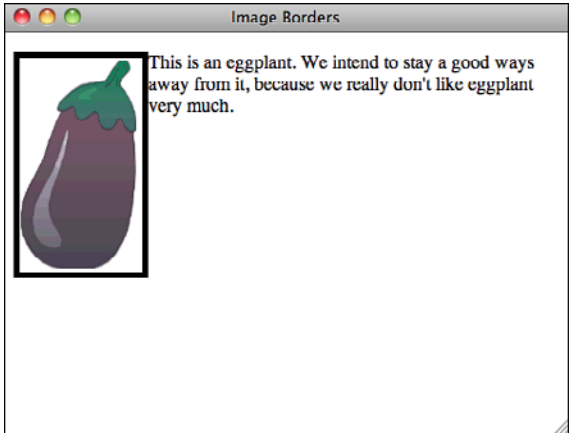


FIGURE 9.16
An image border.

You can also use any of the CSS border styles to borders to an image. In HTML5, the `border` attribute has been removed.

Using Color

As you've seen, one way to add a splash of color to the black, gray, and white on your web pages is to add images. Another is to change the colors of various elements on the page. At one time, HTML attributes were used to specify the colors for various attributes on web pages, but these days, colors should be specified using CSS.

Specifying Colors

Before you can change the color of any part of an HTML page, you have to know what color you're going to change it to. There are three ways to specify colors using CSS, but only two of them work when you're using HTML.

- Using a hexadecimal number representing that color
- Specifying the shades of red, green, and blue as numbers or percentages
- Using one of a set of predefined color names

The most flexible and widely supported method of specifying a color is to use the hexadecimal identifier. Most image-editing programs have what's called a *color picker*—a tool for choosing a single color from a range of available colors. Most color pickers can be configured to display the value of that color in RGB form as three numbers representing the intensity of red, green, and blue in that color. Each number is usually 0 to 255, with `0, 0, 0` being black and `255, 255, 255` being white. If you use one of these tools, you'll have to convert the decimal numbers to hexadecimal. These days, most tools with color pickers also provide the hexadecimal values for red, green, and blue, which is what web browsers require. The color picker that's built in to the Mac OS includes the hexadecimal values to make things easy on web publishers.

If you want to use the regular decimal numbers, you can do so in CSS using the following format:

```
rgb(0,0,0) /* Black */

rgb(255,255,255) /* White */

rgb(255,0,0) /* Red */
```

If you prefer, you can also use percentages:

```
rgb(0%,0%,0%) /* Black */
rgb(100%,100%,100%) /* White */
rgb(100%,0%,0%) /* Red */
```

To specify the colors in the hexadecimal format, you merge the hexadecimal equivalents into a single number, like this:

```
#000000 /* Black */

#FFFFFF /* White */

#FF0000 /* Red */

#660099 /* Purple */
```


Browsers also support specifying colors by name. Instead of using numbers, you just choose a color name such as Black, White, Green, Maroon, Olive, Navy, Purple, Gray, Red, Yellow, Blue, Teal, Lime, Aqua, Fuchsia, or Silver.

Although color names are easier to read and remember than the numbers, only a few colors have names that are well supported by web browsers. Most people tend to use one of the two numeric formats when specifying colors.

There are also a number of websites that are designed to help web designers choose colors. One of the best is Color Schemer at http://www.colorschemer.com/online.html. It enables you to view several colors next to each other to see how they match, and will even suggest colors that match the ones you choose. The current Color Schemer interface appears in Figure 9.17.

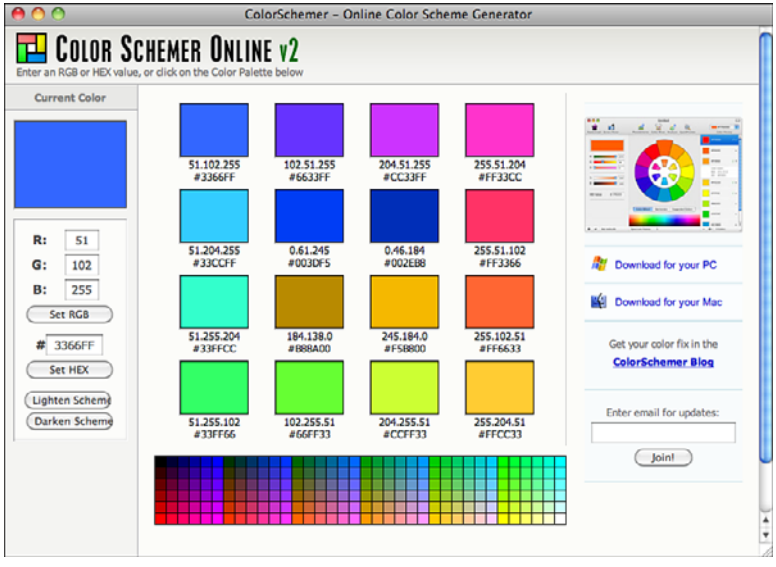


FIGURE 9.17
Color Schemer.

Another is Adobe Kuler at http://kuler.adobe.com/. It makes it easy to create your own color schemes and to browse and rate color schemes created by others. It's a great place to find inspiration if you're thinking about adding color to a site.

Changing Colors Using CSS

There are two key properties when it comes to assigning colors to elements using CSS—`color` and `background-color`. For elements with borders, you can also set the border color using the `border-color` property.

To indicate that a paragraph should have white text on a black background, you could use the following code:

```
<p style="color: #fff, background-color: #000">This paragraph has
white text on a black background.</p>
```

You can also use these properties to adjust the colors on the whole page by applying them to the body tag. Here's an example:

```
<body style="color: #fff; background-color: #00f">
```

This page will have white text on a blue background. You can also specify colors as part of the `background` and `border` properties, which allow you to set the values for a whole family of properties at once. The `background` property will be discussed later in this lesson.

Changing Page Colors in HTML

Before CSS was introduced, page colors were changed using attributes for the `<body>` tag. Many web developers still use these attributes, but they have been deprecated since HTML 4 and were left out of HTML5 entirely. The values of all the color-related HTML attributes can be specified using hexadecimal notation or with color names.

To set the background color for a page, use the `bgcolor` attribute. Here's how it is used with hexadecimal color specifications:

```
<body bgcolor="#ffffff">
<body bgcolor="#934ce8">
```

To use color names, just use the name of the color as the value to `bgcolor`:

```
<body bgcolor="white">
<body bgcolor="green">
```

There are also attributes of the `<body>` tag that can be used to set the text colors for the page. The attributes and their use are as follows:

`text`	Controls the color of all the page's body text except for link text, including headings, body text, text inside tables, and so on.

`link`	Controls the color of link text for links that the user has not already clicked on.
`vlink`	Controls the color of links that the user has already visited.
`alink`	Controls the color of a link while the user is clicking on it. When the user clicks on a link, it changes to this color. When he releases the mouse button, it switches back.

Like the `bgcolor` attribute, these attributes have been dropped from HTML5. You may still see them in older pages, but you should use CSS when you're creating your own pages.

Remember the haunted house image that you inserted on a page earlier? The page would be decidedly spookier with a black background, and orange text would be so much more appropriate for the holiday. To create a page with a black background, orange text, and deep red unvisited links, you might use the following `<body>` tag:

```
<body bgcolor="#000000" text="#ff9933" link="#800000">
```

Using the following color names for the background and unfollowed links would produce the same effect:

```
<body bgcolor="orange" text="black" link="#800000">
```

Both these links would produce a page that looks something like the one shown in Figure 9.18.

FIGURE 9.18
Background and text colors.

The same effect can be achieved using the `background-color` and `color` properties in CSS. Here's the <body> tag that specifies the color using CSS:

```
<body style="background-color: #000000; color: #ff9933">
```

That `style` attribute does not specify a color for links as the original body tag does. To set the link color for all the links on a page, you need to use a style sheet for the page and specify the style for the <a> tag, like this:

```
<style type="text/css">
    a { color: #ff9933; }
</style>
```

What about active links and visited links? CSS provides pseudo-selectors that apply to links in particular states, as follows:

`a:link`	Applies to unvisited links.
`a:visited`	Applies to links which the user has visited.
`a:hover`	Applies to links when the user has her mouse pointer over the link.
`a:active`	Like the `alink` attribute, this selector is used when the user is clicking on the link.

As you can see, these selectors provide access to an additional state that the old attributes did not, the hover state. Here's an example that specifies the colors for links in each of their states:

```
<style type="text/css">
    a { color: #ff9933; }
    a:visited { color: #bbbbbb; }
    a:hover { color: #E58A2E; }
    a:active { color: #FFA64D; }
</style>
```

Image Backgrounds

The last topic in this lesson is using an image as a background for your pages, rather than simply a solid-colored background. When you use an image for a background, that image is *tiled*; that is, it's repeated in rows to fill the browser window.

To create a tiled background, you need an image to serve as the tile. Usually, when you create an image for tiling, you must make sure that the pattern flows smoothly from one tile to the next. You can do some careful adjusting of the image in your favorite image-editing program to make sure that the edges line up. The goal is for the edges to meet cleanly so that you don't have a seam between the tiles after you've laid them end to end. (See 9.19 for an example of tiles that don't line up very well.) You also can try clip art packages for wallpaper or tile patterns that are designed as tiles. Some graphics packages, such as Photoshop, can also modify your images so that they work as tiles. This feature works better with some kinds of images than others.

You can include background images on your pages using CSS. To include a background image on a page (or under any block element), the `background-image` property is used. Here's an example, the results of which are shown in Figure 9.19:

```
<body style="background-image: url(backgrounds/rosemarble.gif)">
```

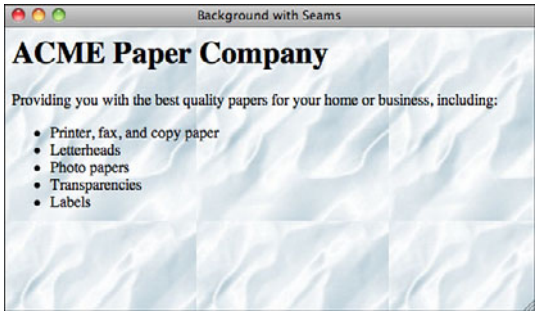


FIGURE 9.19
Tiled images with seams.

You can also use the `background` attribute of the `<body>` tag to specify an image to use as the page background, however, this attribute is not valid in HTML5:

```
<body background="backgrounds/rosemarble.gif" />
```

You may still see the `background` attribute used in older web pages, but you should use CSS for specifying backgrounds.

By default, background images are tiled both horizontally and vertically. However, CSS provides a number of options for controlling how backgrounds are applied. The `background-repeat` property is used to specify how background images are tiled. Options include `repeat` (which tiles the image horizontally and vertically), `repeat-x` (tile horizontally only), `repeat-y` (tile vertically only), and `no-repeat`. You can also specify whether the background image scrolls along with the content of the page or remains in a fixed position using the `background-attachment` property. The two values there are `scroll` and `fixed`. So, if you want to put one background image in the upper-left corner of the browser window and have it stay there as the user scrolls down the page, you would use the following:

```
<body style="background-image: url(backgrounds/rosemarble.gif);
background-repeat: no-repeat; background-attachment: fixed">
```

What if you want the background image to appear somewhere on the page other than the upper-left corner? The `background-position` property enables you to position a background image anywhere you like within a page (or element). The `background-position` property is a bit more complex than most you'll see. You can either pass in two percent-

ages, or the horizontal position (`left`, `right`, `center`), or the vertical position (`top`, `bottom`, `center`) or both the horizontal and vertical position. Here are some valid settings for this property:

Upper right	`0% 100%`
	`top right`
	`right top`
	`right`
Center	`50% 50%`
	`center center`
Bottom center	`50% 100%`
	`bottom center`
	`center bottom`

Here's a `<body>` tag that places the background in the center right of the window and does not scroll it with the page:

```
<body style="background-image: url(backgrounds/rosemarble.gif);
background-repeat: no-repeat;
background-attachment: fixed;
background-position: center right">
```

Instead of using all these different properties to specify the background, you can use the `background` property by itself to specify all the background properties. With the `background` property, you can specify the background color, image, repeat setting, attachment, and position. All the properties are optional, but they must appear in a specific order. Here's the structure of the property:

```
background: background-color background-image background-repeat background-
attachment background-position;
```

To condense the preceding specification into one property, the following tag is used:

```
<body style="background: url(backgrounds/rosemarble.gif)
no-repeat fixed center right">
```

If you like, you can also include a background color. Here's what the new tag looks like:

```
<body style="background: #000 url(backgrounds/rosemarble.gif)
no-repeat fixed center right">
```

Specifying Backgrounds for Elements

The CSS background properties can be used to apply a background to any block-level element—a `<div>`, a `<form>`, a `<p>`, a `<table>`, or a table cell. Specifying the backgrounds for these elements is the same as setting the background for an entire page, and is an easy way to add decorative elements to a page. For example, I want to add an arrow to the left of all the major headings on my web page to really emphasize them. I could stick an `<img>` tag inside each of the `<h1>` elements on my page, but that would be repetitive, and the images don't make sense as part of the contents of the page. So instead, I can apply the images using CSS. The results are in Figure 9.20.

Here's the style sheet:

```
<style type="text/css">
    h1 {
         background: url(arrow_right.png) no-repeat center left;
        line-height: 60px;
         padding-left: 60px;
    }
</style>
```

The heading tag is completely normal:

```
<h1>This Heading Has a Background</h1>
```

FIGURE 9.20
A heading with an image background.

In the style sheet, I use the `background` property to specify the background image and its position for the `<h1>` tag. One extra benefit is that this style will be applied to all the `<h1>` tags on the page. I use the `background` property to specify the image and to position it. I use `no-repeat` because I want only one copy of the image and `center left` to position it at the left end of the header, in the middle. I use the `line-height` property to make sure there's plenty of space for my background image, which is 50 pixels tall, regardless of the font size of the heading. Then I add 60 pixels of padding to the left of the heading so that the text will not appear over the background image.

CSS Backgrounds and the `<img>` Tag

Applying backgrounds to elements using CSS is an alternative to using the `<img>` tag to place images on a page. There are many situations in which both options will work. (For example, if you want to layer text over an image, you can place the text in a `<div>` and use the image as the background for that `<div>`, or you can use the `<img>` tag and then use CSS positioning to place the text over the image.)

However, there is a rule of thumb that many web designers use when choosing between the two alternatives. If an image is purely decorative, it should be included on the page as a background. If an image is part of the content of the page, you should use an `<img>` tag. So if the page is a news article, and you're including an image to illustrate the article, the `<img>` tag is appropriate. If you have a photo of a landscape that you want to use to make the heading of your page more attractive, it makes more sense to use CSS to include the image as a background in the heading.

The reason for this rule is that it makes things easier for visually challenged users who may be visiting a page using a screen reader. If you include your pretty header image using the `<img>` tag, their screen reader will tell them about the image on every page they visit. On the other hand, they probably would want to know about the photo accompanying a news article.

Another simple rule is to think about what you would put in the `alt` attribute for an image. If the alternate text is interesting or useful, you should use the `<img>` tag. If you can't think of anything interesting to put in the alternate text for an image, it probably should be a background for an element instead.

Using Images As Bullets

In Lesson 5, "Organizing Information with Lists," I discussed the `list-style-image` property, which enables you to use images as bullets for lists. Specifying the image URL for bullets is the same as specifying the URL for background images in CSS. The browser will substitute the default bullets with the image you specify. Here's an example, the results of which are shown in Figure 9.21.

Input ▼

```
<!DOCTYPE html>
<html>
  <head>
    <title>Southern Summer Constellations</title>
    <style type="text/css" media="screen">
      ul {
```

```
        list-style-image: url("Bullet.png");
      }
    </style>
  </head>
  <body>
    <h1>Southern Summer Constellations</h1>

    <ul>
      <li>Canis Major</li>
      <li>Cetus</li>
      <li>Eridanus</li>
      <li>Gemini</li>
      <li>Orion</li>
      <li>Perseus</li>
      <li>Taurus</li>
    </ul>
  </body>
</html>
```

Output ▶

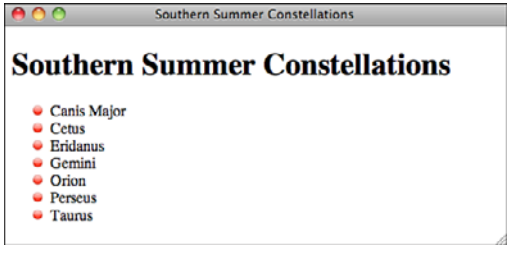


FIGURE 9.21
A list that uses images for bullets.

You can also supply both the `list-style-image` and `list-style-type` properties so that if the image is not found, the list will use the bullet style of your choosing.

What Is an Imagemap?

Earlier in this lesson, you learned how to create an image that doubles as a link simply by including the `<img>` tag inside a link tag (`<a>`). In this way, the entire image becomes a link.

In an imagemap, you can define regions of an image as links. You can specify that certain areas of a map link to various pages, as in Figure 9.22. Or you can create visual metaphors for the information you're presenting, such as a set of books on a shelf or a photograph with a link from each person in the picture to a page with his or her biography on it.

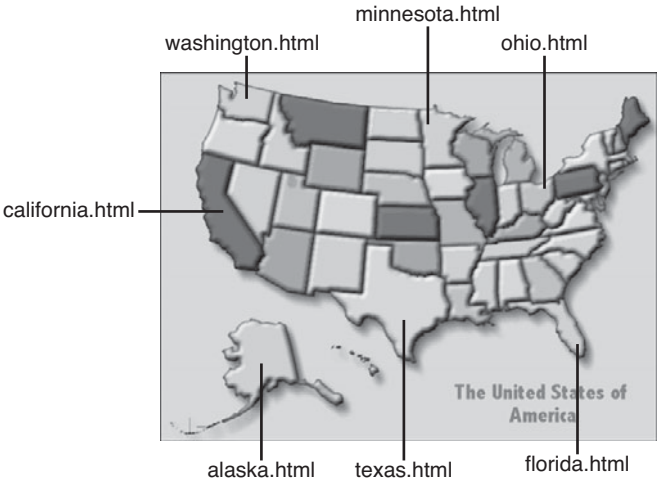


FIGURE 9.22
Imagemaps: different places, different links.

HTML5 supports the `<map>` element for creating image maps. If you don't want to use the `<map>` tag, you can also use CSS to position links over an image and hide the contents of those links, making it appear as though regions of an image are clickable. I discuss both techniques in this lesson.

ImageMaps and Text-Only Browsers

Because of the inherently graphical nature of image maps, they work well only in graphical browsers. Lynx, the most popular text-based browser, provides limited support for client-side imagemaps. If you load a page in Lynx that contains a client-side imagemap, you can get a list of the links contained in the imagemap.

Getting an Image

To create an imagemap, you need an image (of course). This image will be the most useful if it has several discrete visual areas that can be selected individually. For example, use an image that contains several symbolic elements or that can be easily broken down into polygons. Photographs generally don't make good imagemaps because their various elements tend to blend together or are of unusual shapes. Figures 9.23 and 9.24 show examples of good and poor images for imagemaps.

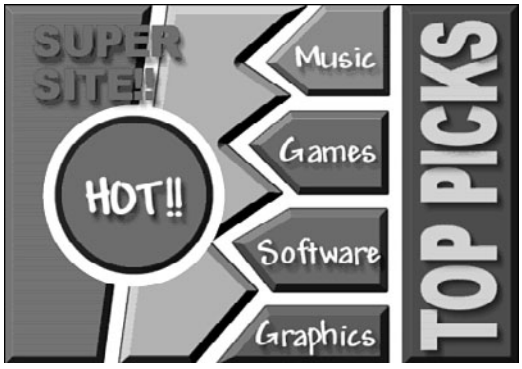


FIGURE 9.23
A good image for an imagemap.

FIGURE 9.24
A not-so-good image for an imagemap.

Determining Your Coordinates

Client-side imagemaps consist of two parts; the first is the image used for the imagemap. The second is the set of HTML tags used to define the regions of the imagemap that serve as links. To define these tags, you must determine the exact coordinates on your image that define the regions you'll use as links.

You can determine these coordinates either by sketching regions and manually noting the coordinates or by using an imagemap creation program. The latter method is easier because the program automatically generates a map file based on the regions you draw with the mouse.

The Mapedit program for Windows, Linux, and the Mac OS can help you create client-side imagemaps. You can find it online at http://www.boutell.com/mapedit/. In addition, many of the latest WYSIWYG editors for HTML pages and Web graphics enable you to generate imagemaps. There's a Web-based editor for imagemaps that you can try out at http://www.image-maps.com/; it creates both imagemaps and the CSS equivalents.

If you must create your imagemaps by hand, here's how. First, make a sketch of the regions that will be active on your image. Figure 9.25 shows the three types of shapes that you can specify in an imagemap: circles, rectangles, and polygons.

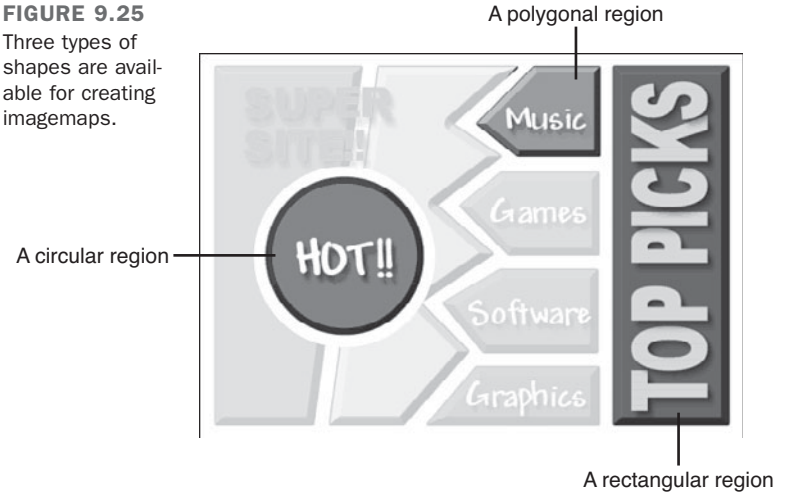


FIGURE 9.25
Three types of shapes are available for creating imagemaps.

You next need to determine the coordinates for the endpoints of those regions. Most image-editing programs have an option that displays the coordinates of the current mouse position. Use this feature to note the appropriate coordinates. (All the mapping programs mentioned previously will create the `<map>` tag for you, but for now, following the steps manually will help you better understand the processes involved.)

Defining a Polygon

Figure 9.26 shows the x,y coordinates of a polygon region. These values are based on their positions from the upper-left corner of the image, which is coordinate 0,0. The first number in the coordinate pair indicates the x value and defines the number of pixels from the extreme left of the image. The second number in the pair indicates the y measurement and defines the number of pixels from the top of the image.

NOTE

The 0,0 origin is in the upper-left corner of the image, and positive y is down.

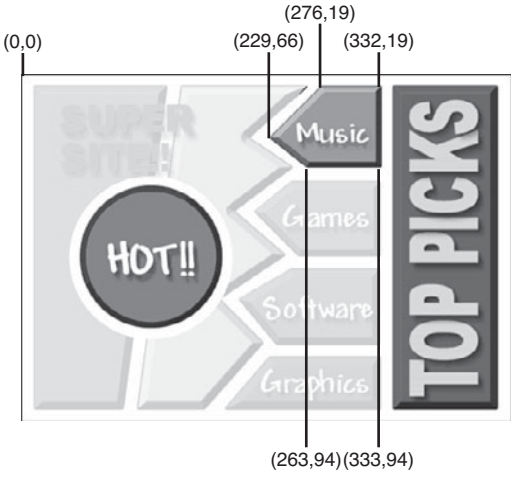


FIGURE 9.26
Getting the coordinates for a polygon.

Defining a Circle

Figure 9.27 shows how to get the coordinates for circles. Here you note the coordinates for the center point of the circle and the radius, in pixels. The center point of the circle is defined as the x,y coordinate from the upper-left corner of the image.

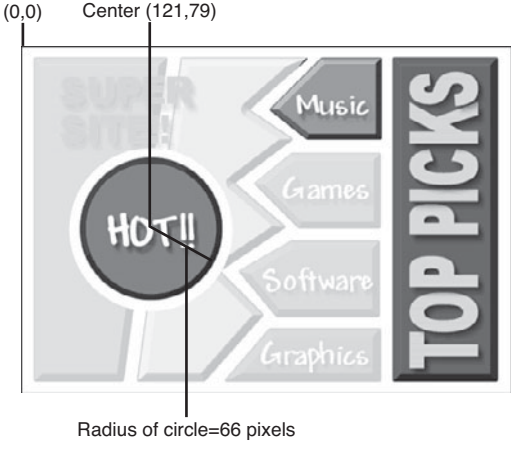


FIGURE 9.27
Getting the coordinates for a circle.

Defining a Rectangle

Figure 9.28 shows how to obtain coordinates for rectangle regions. Note the x,y coordinates for the upper-left and lower-right corners of the rectangle.

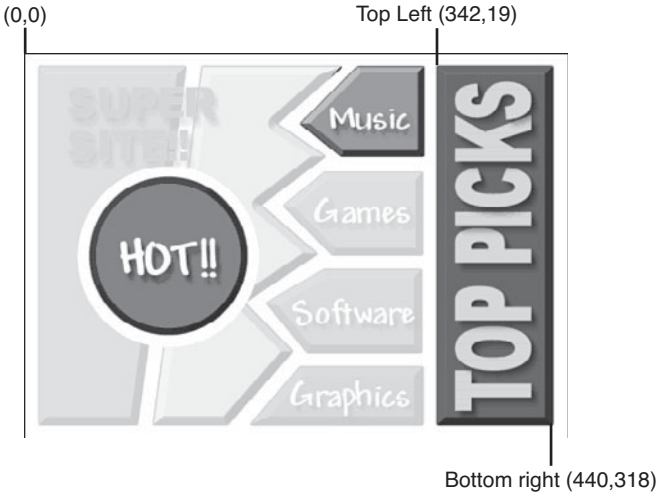


FIGURE 9.28
Getting the coordinates for a rectangle.

The `<map>` and `<area>` Tags

If you're creating your imagemap manually and you've written down all the coordinates for your regions and the URLs they'll point to, you can include this information in the client-side imagemap tags on a web page. To include a client-side imagemap inside an HTML document, use the `<map>` tag, which looks like the following:

```
<map name="mapname"> coordinates and links </map>
```

The value assigned to the `name` attribute is the name of this map definition. This is the name that will be used later to associate the clickable image with its corresponding coordinates and hyperlink references. So, if you have multiple imagemaps on the same page, you can have multiple `<map>` tags with different names.

Between the `<map>` and the `</map>` tags, enter the coordinates for each area in the imagemap and the destinations of those regions. The coordinates are defined inside yet another new tag: the `<area>` tag. To define a rectangle, for example, you would write the following:

```
<area shape="rect" coords="41,16,101,32" href="test.html">
```

The type of shape to be used for the region is declared by the `shape` attribute, which can have the values `rect`, `poly`, `circle`, and `default`. The coordinates for each shape are

noted using the `coords` attribute. For example, the `coords` attribute for a `poly` shape appears as follows:

```
<area shape="poly" coords="x1,y1,x2,y2,x3,y3,...,xN,yN" href="URL">
```

Each *x,y* combination represents a point on the polygon. For `rect` shapes, *x1,y1* is the upper-left corner of the rectangle, and *x2,y2* is the lower-right corner:

```
<area shape="rect" coords="x1,y1,x2,y2" href="URL">
```

For `circle` shapes, *x,y* represents the center of a circular region of size *radius*:

```
<area shape="circle" coords="x,y,radius" href="URL">
```


The `default` shape is different from the others—it doesn't require any coordinates to be specified. Instead, the link associated with the `default` shape is followed if the user clicks anywhere on the image that doesn't fall within another defined region.

Another attribute you need to define for each `<area>` tag is the `href` attribute. You can assign `href` any URL you usually would associate with an `<a>` link, including relative pathnames. In addition, you can assign `href` a value of `"nohref"` to define regions of the image that don't contain links to a new page.

NOTE

If you're using client-side imagemaps with frames, you can include the `target` attribute inside an `<area>` tag to open a new page in a specific window, as in this example:

```
<area shape="rect" coords="x1,y1,x2,y2" href="URL" target=
"window_name">
```

You need to include one more attribute in HTML5. Earlier in this lesson, you learned how to specify alternate text for images. In HTML5, the `alt` attribute is an additional requirement for the `<area>` tag that displays a short description of a clickable area on a client-side imagemap when you pass your cursor over it. Using the `<area>` example that I cited, the `alt` attribute appears as shown in the following example:

```
<area shape="rect" coords="41,16,101,32" href="test.html" alt="test link">
```

The `usemap` Attribute

After you've created your `<map>` tag and defined the regions of your image using `<area>` tags, the next step is to associate the map with the image. To do so, the `usemap` attribute of the `<img>` tag is used. The map name that you specified using the `name` attribute of the `<map>` tag, preceded by a `#`, should be used as the value of the `usemap` attribute, as shown in this example:

```
<img src="image.gif" usemap="#mapname">
```

NOTE

The value assigned to `usemap` is a standard URL. This is why *`mapname`* has a pound symbol (#) in front of it. As with links to anchors inside a web page, the pound symbol tells the browser to look for *`mapname`* in the current web page. If you have a complex imagemap, however, you can store it in a separate HTML file and reference it using a standard URL.

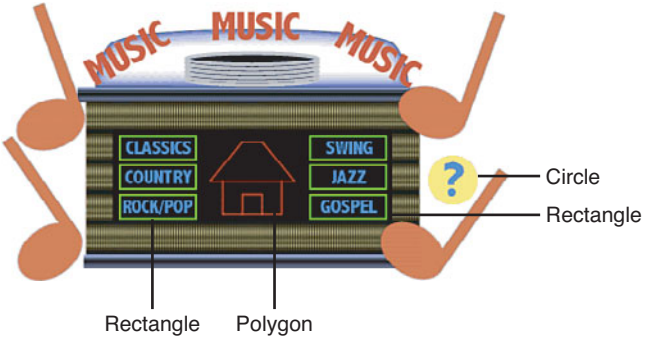
Task: Exercise 9.3: A Clickable Jukebox

Let's take a look at how to create a client-side imagemap for a real image. In this example, you define clickable regions on an image of a jukebox. The image you use appears in Figure 9.29.

FIGURE 9.29
The jukebox image.

First, define the regions that will be clickable on this image. There are six rectangular buttons with musical categories on them, a center area that looks like a house, and a circle with a question mark inside it. Figure 9.30 shows regions on the image.

FIGURE 9.30
The jukebox with areas defined.



Now that you know where the various regions are, you need to find the exact coordinates of the areas as they appear in your image. You can use a mapping program like Mapedit, or you can do it manually. If you try it manually, it's helpful to keep in mind that most image-editing programs display the x and y coordinate of the image when you move the mouse over it.

Getting Image Coordinates from the Browser

You don't have an image-editing program? If you use Firefox as your browser, here's a trick: Create an HTML file with the image inside a link pointing to a fake file, and include the `ismap` attribute inside the `<img>` tag. You don't need a real link; anything will do. The HTML code might look something like the following:

```
<a href="nothing"><img src="myimage.gif" ismap></a>
```

When you load this into your browser, the image displays as if it were an imagemap. When you move your mouse over it, the x and y coordinates appear in the status line of the browser. Using this trick, you can find the coordinates for the map file of any point on that image.

With regions and a list of coordinates, all you need are the web pages to jump to when the appropriate area is selected. These can be documents, scripts, or anything else you can call from a browser as a jump destination. For this example, I've created several documents and stored them inside the `music` directory on my Web server. These are the pages you'll define as the end points when the clickable images are selected. Figure 9.31 identifies each of the eight clickable areas in the imagemap. Table 9.1 shows the coordinates of each and the URL that's called up when it's clicked.

TABLE 9.1 Clickable Areas in the Jukebox Image

Number	Type	URL	Coordinates
1	`rect`	music/classics.html	101,113,165,134
2	`rect`	music/country.html	101,139,165,159
3	`rect`	music/rockpop.html	101,163,165,183
4	`poly`	music/home.html	175,152,203,118
			220,118,247,152
			237,153,237,181
			186,181,186,153

TABLE 9.1 Continued

Number	Type	URL	Coordinates
5	rect	music/swing.html	259,113,323,134
6	rect	music/jazz.html	259,139,323,159
7	rect	music/gospel.html	259,163,323,183
8	circle	music/help.html	379,152,21

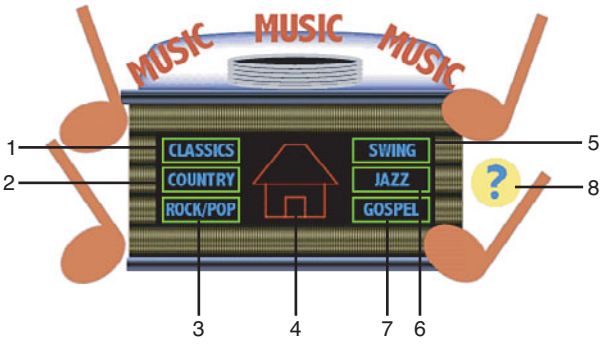


FIGURE 9.31
Eight hot spots, numbered as identified in Table 9.1.

For the jukebox image, the <map> tag and its associated <area> tags and attributes look like the following:

```
<map name="jukebox">
<area shape="rect" coords="101,113, 165,134"
  href="/music/classics.html"
  alt="Classical Music and Composers" />
<area shape="rect" coords="101,139, 165,159"
  href="/music/country.html"
  alt="Country and Folk Music" />
<area shape="rect" coords="101,163, 165,183"
  href="/music/rockpop.html"
  alt="Rock and Pop from 50's On" />
<area shape="poly" coords="175,152, 203,118, 220,118, 247,152,
  237,153, 237,181, 186,181, 186,153"
  href="code/music/home.html"
  alt="Home Page for Music Section" />
<area shape="rect" coords="259,113, 323,134"
  href="/music/swing.html"
  alt="Swing and Big Band Music" />
<area shape="rect" coords="259,139, 323,159"
  href="/music/jazz.html"
  alt="Jazz and Free Style" />
<area shape="rect" coords="259,163, 323,183"
  href="/music/gospel.html"
  alt="Gospel and Inspirational Music" />
```

```
<area shape="circle" coords="379,152, 21"
  href="/music/help.html"
  alt="Help" />
</map>
```

The <img> tag that refers to the map coordinates uses usemap, as follows:

```
<img src="jukebox.gif" usemap="#jukebox">
```

Finally, put the whole thing together and test it. Here's a sample HTML file for The Really Cool Music Page with a client-side imagemap, which contains both the <map> tag and the image that uses it:

Input ▼

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>The Really Cool Music Page</title>
</head>
<body>
<div align="center">
<h1>The Really Cool Music Page</h1>
<p>Select the type of music you want to hear.<br />
 You'll go to a list of songs that you can select from.</p>
<p>
<img src="jukebox.gif" alt="Juke Box" usemap="#jukebox" />
<map name="jukebox">
<area shape="rect" coords="101,113, 165,134"
  href="/music/classics.html"
  alt="Classical Music and Composers" />
<area shape="rect" coords="101,139, 165,159"
  href="/music/country.html"
  alt="Country and Folk Music" />
<area shape="rect" coords="101,163, 165,183"
  href="/music/rockpop.html"
  alt="Rock and Pop from 50's On" />
<area shape="poly" coords="175,152, 203,118, 220,118, 247,152,
  237,153, 237,181, 186,181, 186,153"
  href="code/music/home.html"
  alt="Home Page for Music Section" />
<area shape="rect" coords="259,113, 323,134"
  href="/music/swing.html"
  alt="Swing and Big Band Music" />
<area shape="rect" coords="259,139, 323,159"
  href="/music/jazz.html"
  alt="Jazz and Free Style" />
<area shape="rect" coords="259,163, 323,183"
```

```
    href="/music/gospel.html"
    alt="Gospel and Inspirational Music" />
<area shape="circle" coords="379,152, 21"
    href="/music/help.html"
    alt="Help" />
</map></p>
<p>
<a href="code/music/home.html">Home</a> ¦
<a href="code/music/classics.html">Classics</a> ¦
<a href="code/music/country.html">Country</a> ¦
<a href="code/music/rockpop.html">Rock/Pop</a> ¦
<a href="code/music/swing.html">Swing</a> ¦
<a href="code/music/jazz.html">Jazz</a> ¦
<a href="code/music/gospel.html">Gospel</a> ¦
<a href="code/music/help.html">Help</a>
</p>
</div>
</body>
</html>
```

Figure 9.32 shows the imagemap in a browser.

Output ▶

FIGURE 9.32
The finished Really Cool Music Page with client-side imagemap.

▲

Image Etiquette

There are great images on sites all over the Web: cool icons, great photographs, excellent line art, and plenty of other graphics, too. You might feel the temptation to link directly to these images and include them on your own pages, or to save them to disk and then use them. There are a number of reasons why it's wrong to do so.

First, if you're linking directly to images on another site, you're stealing bandwidth from that site. Every time someone requests your page, they'll also be issuing a request to the site where the image is posted and downloading the image from there. If you get a lot of traffic, you can cause problems for the remote site.

The second reason is actually a problem regardless of how you use images from other sites. If you don't have permission to use an image on your site, you're violating the rights of the image's creator. Copyright law protects creative work from use without permission, and it's granted to every creative work automatically.

The best course of action is to create your own images or look for images that are explicitly offered for free use by their creators. Even if images are made available for your use, you should download them and store them with your web pages rather than linking to them directly. Doing so prevents you from abusing the bandwidth of the person providing the images.

Summary

In this lesson you learned to place images on your web pages. Those images are normally in GIF, JPEG, or PNG format and should be small enough that they can be downloaded quickly over a slow link. You also learned that the HTML tag `<img>` enables you to put an image on a web page either inline with text or on a line by itself. The `<img>` tag has three primary attributes supported in standard HTML:

`src`	The location and filename of the image to include
`align`	How to position the image vertically with its surrounding text. `align` can have one of three values: `top`, `middle`, or `bottom` (deprecated from HTML5 in favor of style sheets).
`alt`	A text string to substitute for the image in text-only browsers

You can include images inside a link tag (`<a>`) to treat them as links.

In addition to the standard attributes, several other attributes to the `<img>` tag provide greater control over images and layout on web pages. You learned how to use these HTML 3.2 attributes in this lesson, but they have been removed from HTML5 in favor of style sheets. They include the following:

`align="left"`	Places the image against the appropriate margin, `align="right"`, allowing all the following text to flow into the space alongside the image.
`clear`	An extension to ` ` that enables you to stop wrapping text alongside an image. `clear` can have three values: `left`, `right`, and `all`.

`align="texttop"`	Allows greater control over the alignment of an inline `align="absmiddle"` image and the text surrounding it.
`vspace`	Defines the amount of space between an image `hspace` and the text surrounding it.
`border`	Defines the width of the border around an image (with or without a link). `border="0"` hides the border altogether.

In addition to images, you can add color to the background and the text of a page using CSS.You learned the CSS properties `color`, `background-color`, and `background`, which enable you to specify colors for your page without using deprecated tags. You also learned about the HTML attributes that were once used to specify colors before CSS was widely supported.

Workshop

Now that you know how to add images and color to your pages, you can really get creative. This workshop will help you remember some of the most important points about using images and color on your pages..

Q&A

Q What's the difference between a GIF image and a JPEG image? Is there any rule of thumb that defines when you should use one format rather than the other?

A As a rule, you should use GIF files when images contain 256 colors or fewer. Some good examples are cartoon art, clip art, black-and-white images, and images with many solid color areas. You'll also need to use GIF files if you want your images to contain transparent areas or if you want to create an animation that doesn't require a special plug-in or browser helper. Remember to use your image-editing software to reduce the number of colors in the image palettes whenever possible, because this also reduces the size of the file.

JPEG images are best for photographic-quality or high-resolution 3D rendered graphics because they can display true-color images to great effect. Most image-editing programs enable you to specify how much to compress a JPEG image. The size of the file decreases the more an image is compressed; however, compression can also deteriorate the quality and appearance of the image if you go overboard. You have to find just the right balance between quality and file size, and that can differ from image to image.

Q My client-side imagemaps aren't working. What's wrong?

A Make sure that the pathnames or URLs in your `<area>` tags point to real files. Also, make sure the map name in the `<map>` file matches the name of the map in the `usemap` attribute in the `<img>` tag. Only the latter should have a pound sign in front of it.

Q How can I create thumbnails of my images so that I can link them to larger external images?

A You'll have to do that with some type of image-editing program (such as Adobe Photoshop); the Web won't do it for you. Just open up the image and scale it down to the right size.

Q What about images that are partially transparent so that they can display the page background? They look like they sort of float on the page. How do I create those?

A This is another task you can accomplish with an image-editing program. Both GIF and PNG support transparency. Most image-editing programs provide the capability to create these types of images.

Q Can I put HTML tags in the string for the `alt` attribute?

A That would be nice, wouldn't it? Unfortunately, you can't. All you can do is put an ordinary string in there. Keep it simple, and you should be fine.

Quiz

1. What's the most important attribute of the `<img>` tag? What does it do?
2. If you see a funny-looking icon rather than an image when you view your page, the image isn't loading. What are some of the reasons this could happen?
3. Why is it important to use the `alt` attribute to display a text alternative to an image? When is it most important to do so?
4. What is an imagemap?
5. Why is it a good idea to also provide text versions of links that you create on an imagemap?
6. True or false: When you use the `background` shorthand property, the order of the values is important.

Quiz Answers

1. The most important attribute of the `<img>` tag is the `src` attribute. It indicates the filename or URL of the image you want to include on your page.
2. Several things might cause an image not to load: The URL may be incorrect; the filename might not be correct (they're case-sensitive); it might have the wrong file extension; or it might be the wrong type of file.
3. It's a good idea to provide text alternatives with images because some people use text-only browsers or have their graphics turned off. It's especially important to provide text alternatives for images used as links.
4. An imagemap is a special image in which different areas point to different locations on the Web.
5. It's a good idea to include text versions of imagemap links in case there are users who visit your page with text-only browsers or with images turned off. This way, they can still follow the links on the web page and visit other areas of your website.
6. True. The property will work only if you enter the values in the proper order.

Exercises

1. Create or find some images that you can use as navigation icons or buttons on one or more pages of your website. Remember that it's always advantageous to use images more than once. Create a simple navigation bar that you can use on the top or bottom of each page.
2. Create or find some images that you can use to enhance the appearance of your web pages. After you find some that you like, try to create background, text, and link colors that are compatible with them.
3. Create and test a simple client-side imagemap that links to pages that reside in different subdirectories in a website or to other sites on the World Wide Web.
4. Create and test a client-side imagemap for your own home page, or for the entry page in one of the main sections of your website. Remember to include alternatives for those who are using text-only browsers or browsers designed for the disabled.

LESSON 10
Building Tables

So far in this book, you've used plain vanilla *Hypertext Markup Language* (HTML) to build and position the elements on your pages, and you've used Cascading Style Sheets (CSS) to fine-tune their appearance. Although you can get your point across using paragraphs and lists, some information lends itself best to being presented in tables. In this lesson, you learn how to use HTML to create them.

When tables were officially introduced in HTML 3.2, they were commonly used to lay out entire pages. More recently, that role has been taken over by CSS. With the introduction of HTML 4 and later releases, new features were added to enable tables to better perform their designated role: the presentation of tabular data.

In this lesson, you'll learn all about tables, including the following:

- Defining tables in HTML
- Creating captions, rows, and heading and data cells
- Modifying cell alignment
- Creating cells that span multiple rows or columns
- Adding color to tables
- Using tables in web pages

Creating Tables

Creating tables in HTML is a degree more complex than anything you've seen so far in this book. Think about how many different types of tables there are. A table can be a three-by-three grid with labels across the top, or two side-by-side cells, or a complex Excel spreadsheet that comprises many rows and columns of various sizes. Representing tables in HTML is heavy on tags, and the tags can be hard to keep track of when you get going.

The basic approach with table creation is that you represent tabular data in a linear fashion, specifying what data goes in which table cells using HTML tags. In HTML, tables are created from left to right and top to bottom. You start by creating the upper-left cell, and finish with the bottom-right cell. This will all become clearer when you see some actual table code.

Table Parts

Before getting into the actual HTML code to create a table, here are some table-related terms you'll see throughout this lesson:

- The *caption* indicates what the table is about: for example, "Voting Statistics, 1950–1994," or "Toy Distribution Per Room at 1564 Elm St." Captions are optional.
- The *table headings* label the rows, columns, or both. Usually they're in an emphasized font that's different from the rest of the table. They're optional.
- *Table cells* are the individual squares in the table. A cell can contain normal table data or a table heading.
- *Table data* is the values in the table itself. The combination of the table headings and table data makes up the sum of the table.

Figure 10.1 shows a typical table and its parts.

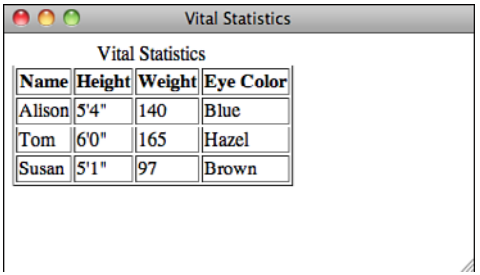


FIGURE 10.1
The elements that make up a table.

The <table> Element

All the components of a table are placed within a <table>...</table> element:

```
<table>
...table caption (optional) and contents...
</table>
```

Here's the code that produces the table shown in Figure 10.1. Don't be concerned if you don't know what this all means right now. For now, notice that the table starts with a <table> tag and its attributes, and ends with a </table> tag:

```
<table border="1">
<caption>Vital Statistics</caption>
 <tr>
  <th>Name</th>
  <th>Height</th>
  <th>Weight</th>
  <th>Eye Color</th>
 </tr>
 <tr>
  <td>Alison</td>
  <td>5'4"</td>
  <td>140</td>
  <td>Blue</td>
 </tr>
 <tr>
  <td>Tom</td>
  <td>6'0"</td>
  <td>165</td>
  <td>Hazel</td>
 </tr>
 <tr>
  <td>Susan</td>
  <td>5'1"</td>
  <td>97</td>
  <td>Brown</td>
 </tr>
</table>
```


The Table Summary

If you want to play by the rules of XHTML, every time you create a table, the <table> element must include the summary attribute. The value of the summary should be a short description of the table's contents. Normal visual browsers don't use this value; instead, it's intended for screen readers and other browsers created for users with disabilities. For example, the <table> tag in the previous example should include a summary attribute like this:

```
<table summary="vital statistics">
```

For your pages to play by the XHTML rules, you must include the summary attribute for all your tables (just as you must include `alt` text for all your images). You'll learn more about why accessibility features are important in Lesson 19, "Designing for the Real World."

Rows and Cells

Now that you've been introduced to the `<table>` element, we'll move on to the rows and cells. Inside the `<table>...</table>` element, you define the actual contents of the table. Tables are specified in HTML row by row, and each row definition contains all the cells in that row. So, to create a table, you start with the top row and then each cell in turn, from left to right. Then you define a second row and its cells, and so on. The number of columns is calculated based on how many cells there are in each row.

Each table row starts with the `<tr>` tag and ends with the closing `</tr>`. Your table can have as many rows and columns as you like, but you should make sure that each row has the same number of cells so that the columns line up.

The cells within each row are created using one of two elements:

- `<th>...</th>` elements are used for heading cells. Generally, browsers center the contents of a `<th>` cell and render any text in the cell in boldface.
- `<td>...</td>` elements are used for data cells. `td` stands for *table data*.

CAUTION

You might have heard somewhere that closing tags are not required for `<th>`, `<td>`, and `<tr>` tags. You might even see HTML that's written without them. However, XHTML requires that you include them, and including them makes your code much easier to follow. Don't leave them out.

In this table example, the heading cells appear in the top row and are defined with the following code:

```
<tr>
 <th>Name</th>
 <th>Height</th>
 <th>Weight</th>
 <th>Eye Color</th>
</tr>
```

The top row is followed by three rows of data cells, which are coded as follows:

```
<tr>
 <td>Alison</td>
 <td>5'4"</td>
 <td>140</td>
 <td>Blue</td>
</tr>
<tr>
 <td>Tom</td>
 <td>6'0"</td>
 <td>165</td>
 <td>Blue</td>
</tr>
<tr>
 <td>Susan</td>
 <td>5'1"</td>
 <td>97</td>
 <td>Brown</td>
</tr>
```


As you've seen, you can place the headings along the top edge by defining the <th> elements inside the first row. Let's make a slight modification to the table. You'll put the headings along the left edge of the table instead. To accomplish this, put each <th> in the first cell in each row, and follow it with the data that pertains to each heading. The new code looks like the following:

Input ▼

```
<tr>
 <th>Name</th>
 <td>Alison</td>
 <td>Tom</td>
 <td>Susan</td>
</tr>
<tr>
 <th>Height</th>
 <td>5'4"</td>
 <td>6'0"</td>
 <td>5'1"</td>
</tr>
<tr>
 <th>Weight</th>
 <td>140</td>
 <td>165</td>
 <td>97</td>
</tr>
<tr>
```

```
 <th>Eye Color</th>
 <td>Blue</td>
 <td>Blue</td>
 <td>Brown</td>
</tr>
```

Figure 10.2 shows how this table displays in a browser.

Output ▶

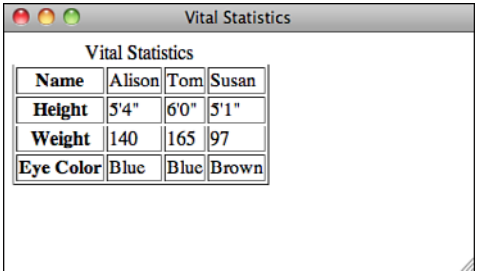


FIGURE 10.2
An example of a table that includes headings in the leftmost column.

Empty Cells

Both table heading cells and data cells can contain any text, HTML code, or both, including links, lists, forms, images, and other tables. But what if you want a cell with nothing in it? That's easy. Just define a cell with a `<th>` or `<td>` element with nothing inside it:

Input ▼

```
<table border="1">
<tr>
  <td></td>
  <td>10</td>
  <td>20</td>
</tr>
</table>
```

Some older browsers display empty cells of this sort as if they don't exist at all. If you want to force a *truly* empty cell, you can add a line break with no other text in that cell by itself:

Input ▼

```
<table border="1">
<tr>
  <td><br /></td>
```

```
  <td>10</td>
  <td>20</td>
</tr>
</table>
```

Figure 10.3 shows examples of both types of empty cells: the empty cell and the really empty cell with the line break added.

Output ▶

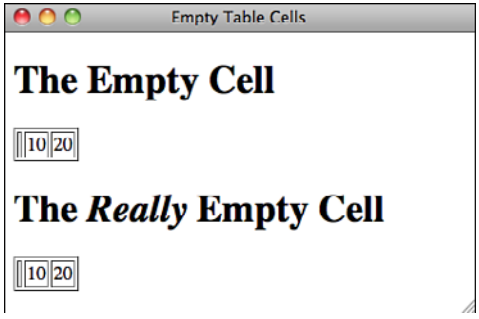


FIGURE 10.3
Empty cells and really empty cells.

Captions

Table captions tell your visitor what the table is for. The `<caption>` element, created just for this purpose, displays the text inside the tag as the table caption (usually centered above the table). Although you could use a regular paragraph or a heading as a caption for your table, tools that process HTML files can extract `<caption>` elements into a separate file, automatically number them, or treat them in special ways simply because they're captions.

NOTE

> If you don't want a caption, it's optional. If you just want a table and don't care about a label, leave the caption off.

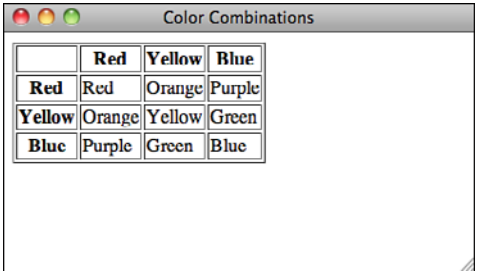
The `<caption>` element goes inside the `<table>` element just before the table rows, and it contains the title of the table. It closes with the `</caption>` tag:

```
<table>
<caption>Vital Statistics</caption>
<tr>
```

Task: Exercise 10.1: Creating a Simple Table

Now that you know the basics of how to create a table, try a simple example. You'll create a table that indicates the colors you get when you mix the three primary colors together. Figure 10.4 shows the table you're going to re-create in this example.

FIGURE 10.4
A simple color table.



Here's a quick hint for laying out tables: Because HTML defines tables on a row-by-row basis, sometimes it can be difficult to keep track of the columns, particularly with complex tables. Before you start actually writing HTML code, it's useful to make a sketch of your table so that you know the heads and the values of each cell. You might even find that it's easiest to use a word processor with a table editor (such as Microsoft Word) or a spreadsheet to lay out your tables. Then, when you have the layout and the cell values, you can write the HTML code for that table. Eventually, if you do this enough, you'll think of these things in terms of HTML tags, whether you want.

Start with a simple HTML framework for a page that contains a table. As with all HTML files, you can create this file in any text editor:

```
<!DOCTYPE html>
<html>
<head>
<title>Colors</title>
</head>
<body>
<table>
...add table rows and cells here...
</table>
</body>
</html>
```

Now start adding table rows inside the opening and closing `<table>` tags (where the line `add table rows and cells here` is). The first row is the three headings along the top of the table. The table row is indicated by `<tr>` and each cell by a `<th>` tag:

```
<tr>
  <th>Red</th>
  <th>Yellow</th>
  <th>Blue</th>
</tr>
```

NOTE

You can format the HTML code any way you want. As with all HTML, the browser ignores most extra spaces and returns. I like to format it like this, with the contents of the individual rows indented and the cell elements on separate lines so that I can pick out the rows and columns more easily.

Now add the second row. The first cell in the second row is the Red heading on the left side of the table, so it will be the first cell in this row, followed by the cells for the table data:

```
<tr>
  <th>Red</th>
  <td>Red</td>
  <td>Orange</td>
  <td>Purple</td>
</tr>
```

Continue by adding the remaining two rows in the table, with the Yellow and Blue headings. Here's what you have so far for the entire table:

Input ▼

```
<table border="1" summary="color combinations">
<tr>
  <th>Red</th>
  <th>Yellow</th>
  <th>Blue</th>
</tr>
<tr>
  <th>Red</th>
  <td>Red</td>
  <td>Orange</td>
  <td>Purple</td>
</tr>
<tr>
  <th>Yellow</th>
  <td>Orange</td>
  <td>Yellow</td>
  <td>Green</td>
```

```
</tr>
<tr>
  <th>Blue</th>
  <td>Purple</td>
  <td>Green</td>
  <td>Blue</td>
</tr>
</table>
```

Finally, add a simple caption. The `<caption>` element goes just after the `<table>` tag and just before the first `<tr>` tag:

```
<table border="1">
<caption>Mixing the Primary Colors</caption>
<tr>
```

With a first draft of the code in place, test the HTML file in your favorite browser that supports tables. Figure 10.5 shows how it looks.

Output ▶

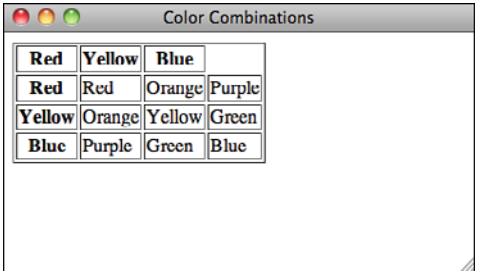


FIGURE 10.5
The not-quite-perfect color table.

Oops! What happened with that top row? The headings are all messed up. The answer, of course, is that you need an empty cell at the beginning of that first row to space the headings out over the proper columns. HTML isn't smart enough to match it all up for you. (This is exactly the sort of error you're going to find the first time you test your tables.)

Add an empty table heading cell to that first row. (Here, it's the line `<th><br /></th>`.)

Input ▼

```
<tr>
  <th><br /></th>
  <th>Red</th>
  <th>Yellow</th>
  <th>Blue</th>
</tr>
```

NOTE

I used <th> here, but it could be <td> just as easily. Because there's nothing in the cell, its formatting doesn't matter.

If you try it again, you should get the right result with all the headings over the right columns, as the original example in Figure 10.4 shows. ▲

Sizing Tables, Borders, and Cells

With the basics out of the way, now you'll look at some of the attributes that can change the overall appearance of your tables. The attributes you'll learn about in this section control the width of your tables and cells, the amount of spacing between cell content and rows and columns, and the width of the borders. As is the case with most attributes relating to formatting, you can also use some CSS properties with the same effect.

Setting Table Widths

The table in the preceding example relied on the browser itself to decide how wide the table and column widths were going to be. In many cases, this is the best way to make sure that your tables are viewable on different browsers with different screen sizes and widths. Just let the browser decide.

In other cases, however, you might want more control over how wide your tables and columns are, particularly if the defaults the browser comes up with are strange. In this section, you'll learn a couple of ways to do just this.

The `width` attribute of the `<table>` element defines how wide the table will be on the page. `width` can have a value that is either the exact width of the table (in pixels) or a percentage (such as 50% or 75%) of the current browser width, which can therefore change if the window is resized. If `width` is specified, the width of the columns within the table can be compressed or expanded to fit the required size.

To make a table as wide as the browser window, you add the `width` attribute to the table, as shown in the following line of code:

Input ▼

```
<table border="1" width="100%">
```

Figure 10.6 shows the result.

Output ▶

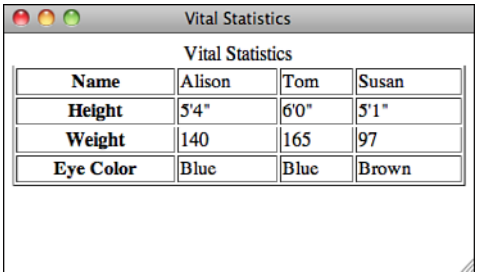


FIGURE 10.6
A table set to 100% width.

CAUTION

If you make your table too narrow for whatever you put in it, the browser will ignore your settings and makes the table as wide as it needs to be to display the content, unless you use the CSS `overflow` property to specify otherwise. The overflow property was discussed in Lesson 8, "Using CSS to Style a Site."

It's nearly always a better idea to specify your table widths as percentages rather than as specific pixel widths. Because you don't know how wide the browser window will be, using percentages allows your table to be reformatted to whatever width the browser is. Using specific pixel widths might cause your table to run off the page. Also, if you make your tables too wide using a pixel width, your pages might not print properly.

NOTE

Specifying column widths in percentages is illegal under the XHTML 1.0 Strict specification. If you want to specify your column widths in that manner, use Transitional DTD or specify the widths in a style sheet. I discuss using style sheets in this manner further along in this lesson.

Changing Table Borders

The `border` attribute, which appears immediately inside the opening `<table>` tag, is the most common attribute of the `<table>` element. With it, you specify whether border lines are displayed around the table and if so, how wide the borders should be.

The `border` attribute has undergone some changes since it first appeared in HTML:

- In HTML 2.0, you used `<table border>` to draw a border around the table. The border could be rendered as fancy in a graphical browser or just a series of dashes and pipes (¦) in a text-based browser.

- Starting with HTML 3.2 and later, the correct usage of the `border` attribute was a little different: It indicates the width of a border in pixels. `<table border="1">` creates a 1-pixel wide border, `<table border="2">` a 2-pixel wide border, and so on. HTML 3.2 and later browsers are expected to display the old HTML 2.0 form of `<table border>`, with no value, with a 2-pixel border (as if you specified `<table border="1">`).
- To create a border that has no width and isn't displayed, you specify `<table border="0">`. Borderless tables are useful when you want to use the table structure for layout purposes, but you don't necessarily want the outline of an actual table on the page. Browsers that support HTML 3.2 and later are expected not to display a border (the same as `<table border="0">`) if you leave out the `border` attribute entirely.

You can change the width of the border drawn around the table. If `border` has a numeric value, the border around the outside of the table is drawn with that pixel width. The default is `border="1"`. `border="0"` suppresses the border, just as if you had omitted the `border` attribute altogether.

Figure 10.7 shows a table that has a border width of 10 pixels. The table and border definition looks like this:

Input ▼

```
<table border="10" width="100%">
```

Output ▶

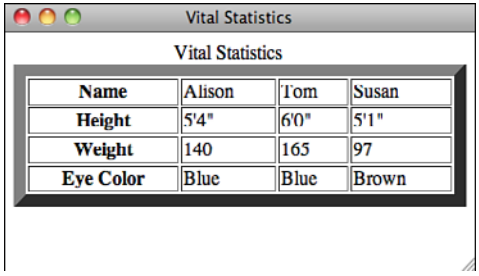


FIGURE 10.7
A table with the border width set to 10 pixels.

You can also adjust the borders around your tables using CSS, with much finer control than the `border` attribute provides.

You learned about borders in Lesson 8, but there's more to them when it comes to tables. For example, if you write a table like the one that follows, it will have a border around the outside but no borders around the cells:

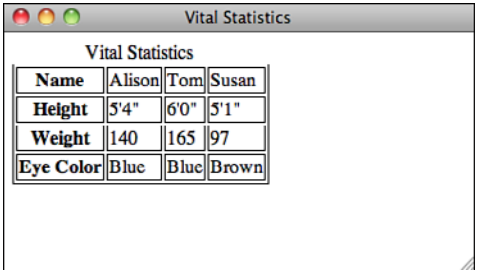
```
<table style="border: 1px solid red">
        <!— Table rows and cells go here. —>
</table>
```

To draw borders around all the cells in a table (the way the `border` attribute does), the easiest way is to use a style sheet like this:

```
<style type="text/css">
table { border: 1px solid black; }
td, th { border: 1px solid black; }
</style>
```

If I applied that style sheet to the color table used in the previous example, it would appear as it does in Figure 10.8.

FIGURE 10.8
A table with cell borders applied using CSS.

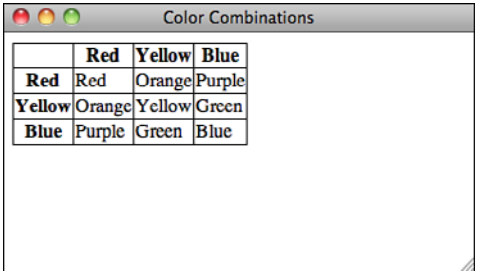


As you can see, there are gaps between the borders on each cell for this table. To fix this, we need to use the CSS `border-collapse` property on the `table` element. It has two possible values, `separate` and `collapse`. The default is `separate`, it produces the result you see in Figure 10.8. The style sheet that follows shows how to apply it:

```
<style type="text/css">
table {
    border: 1px solid black;
    border-collapse: collapse;
}
td, th {
    border: 1px solid black;
}
</style>
```

Figure 10.9 shows the results.

FIGURE 10.9
A table that uses the `border-collapse` property to eliminate space between cells.



NOTE

The table that I used for this example included the `border` attribute to create a border. If you apply table borders using CSS, they will override the `border` attribute, so you don't need to remove it. This can be helpful because primitive browsers (including the browsers on some mobile phones) don't offer CSS support, and including the `border` attribute will ensure that borders are still displayed.

Cell Padding

The cell padding attribute defines the amount of space between the edges of the cells and the content inside a cell. By default, many browsers draw tables with a cell padding of two pixels. You can add more space by adding the `cellpadding` attribute to the `<table>` element, with a value in pixels for the amount of cell padding you want.

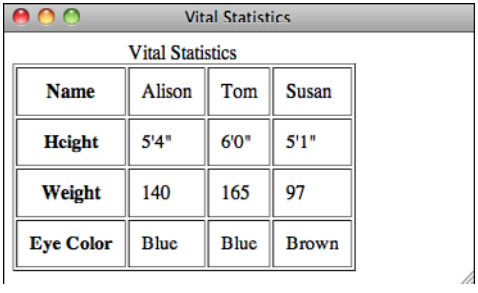
Here's the revised code for your `<table>` element, which increases the cell padding to 10 pixels. The result is shown in Figure 10.10.

Input ▼

```
<table cellpadding="10" border="1">
```

Output ▶

FIGURE 10.10
A table with the cell padding set to 10 pixels.



A `cellpadding` attribute with a value of 0 causes the edges of the cells to touch the edges of the cell's contents. This doesn't look good when you're presenting text, but it can prove useful in other situations.

You can also specify the padding of a table cell using the `padding` property in CSS. The advantages of doing so are that you can specify the padding for the top, left, right, and bottom separately, and that you can specify different padding amounts for different cells of the table if you choose to do so. For example, you can set the padding of header cells to 10 pixels on the top and 5 pixels on the sides and bottom, and then set the padding to four pixels on all for sides for regular table cells.

Cell Spacing

Cell spacing is similar to cell padding except that it affects the amount of space between cells—that is, the width of the space between the inner and outer lines that make up the table border. The `cellspacing` attribute of the `<table>` element affects the spacing for the table. Cell spacing is two pixels by default.

Cell spacing also includes the outline around the table, which is just inside the table's border (as set by the `border` attribute). Experiment with it, and you can see the difference. For example, Figure 10.11 shows our table with cell spacing of 8 and a border of 4, as shown in the following code:

Input ▼

```
<table cellpadding="10" border="4" cellspacing="8">
```

Output ▶

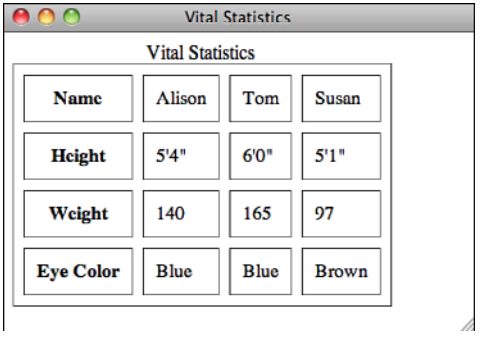


FIGURE 10.11
How increased cell spacing looks.

NOTE

If you want to completely eliminate any whitespace separating content in table cells, you must set the table's border, cell padding, and cell spacing to `0`. Laying out your tables this way is unusual, but it can be useful if you've sliced up an image and you want to reassemble it properly on a web page.

The CSS equivalent of the `cellspacing` attribute is the `border-spacing` property, which must be applied to the table. To use it, the `border-collapse` property must not be set to collapse, as it eliminates cell spacing. `border-spacing` is slightly different than `padding`. With `padding`, you can specify the padding for all four sides of an element. `border-spacing` takes one or two values. If one value is specified, it is used for all four sides of each cell. If two are specified, the first sets the horizontal spacing and the second sets the vertical spacing. The table in Figure 10.12 uses the following style sheet, which sets the cell padding for each cell to 5 pixels, and sets the cell spacing for the table to 10 pixels horizontally and 5 pixels vertically:


```
<style type="text/css">
table {
        border-collapse: separate;
        border-spacing: 10px 5px;
}

td, th {
    border: 1px solid black;
        padding: 5px;
}
</style>
```

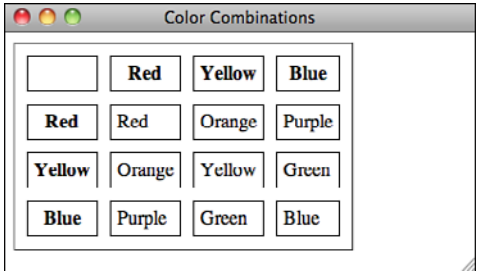


FIGURE 10.12
Using CSS to specify cell spacing and cell padding.

Column Widths

You also can apply the `width` attribute to individual cells (`<th>` or `<td>`) to indicate the width of columns in a table. As with table widths, discussed earlier, you can make the `width` attribute in cells an exact pixel width or a percentage (which is taken as a

percentage of the full table width). As with table widths, using percentages rather than specific pixel widths is a better idea because it allows your table to be displayed regardless of the window size.

Column widths are useful when you want to have multiple columns of identical widths, regardless of their contents (for example, for some forms of page layout).

Figure 10.13 shows your original table from Figure 10.1. This time, however, the table spans 100% of the screen's width. The first column is 40% of the table width, and the remaining three columns are 20% each.

To accomplish this, the column widths are applied to the heading cells as follows:

Input ▼

```
<table border="1" width="100%">
<caption>Vital Statistics</caption>
<tr>
  <th width="40%">Name</th>
  <th width="20%">Height</th>
  <th width="20%">Weight</th>
  <th width="20%">Eye Color</th>
 </tr>
</table>
```

Output ▶

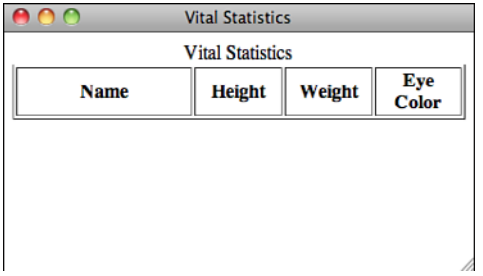


FIGURE 10.13
A table with manually set column widths.

What happens if you have a table that spans 80% of the screen, and it includes the same header cells (40%, 20%, 20%, and 20%) as in the preceding example? Revise the code slightly, changing the width of the entire table to 80%, as shown in Figure 10.14. When you open the new table in your browser, you'll see that the table now spans 80% of the width of your screen. The four columns still span 40%, 20%, 20%, and 20% of the *table*. To be more specific, the columns span 32%, 16%, 16%, and 16% of the entire screen width:

Input ▼

```
<table border="1" width="80%">
<caption>Vital Statistics</caption>
<tr>
  <th width="40%">Name</th>
  <th width="20%">Height</th>
  <th width="20%">Weight</th>
  <th width="20%">Eye Color</th>
 </tr>
</table>
```

Output ▶

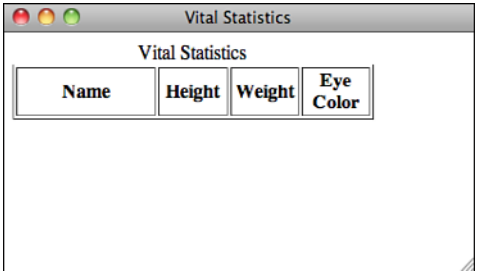


FIGURE 10.14
A modified table with manually set column widths.

If you are going to specify cell widths, make sure to either specify the widths for cells only on one row or to the same values for every row. If you specify more than one value for the width of a column (by specifying different values on multiple rows of a table), there's no good way to predict which one the browser will use.

You can also specify widths using CSS, using the `width` property. Here's how you create the preceding table using CSS for the cell widths:

```
<table border="1" style="width: 80%">
<caption>Vital Statistics</caption>
<tr>
  <th style="width: 40%">Name</th>
  <th style="width: 20%">Height</th>
  <th style="width: 20%">Weight</th>
  <th style="width: 20%">Eye Color</th>
 </tr>
</table>
```

One advantage of using CSS approach is that you can use any units you like to specify the width, rather than just pixels or percentages.

Setting Breaks in Text

Often, the easiest way to make small changes to how a table is laid out is by using line breaks (
 elements). Line breaks are particularly useful if you have a table in which most of the cells are small and only one or two cells have longer data. As long as the screen width can handle it, generally the browser just creates really long rows. This looks rather funny in some tables. For example, the last row in the table shown in Figure 10.15 is coded as follows:

Input ▼

```
<tr>
  <td>TC</td>
  <td>7</td>
  <td>Suspicious except when hungry, then friendly</td>
 </tr>
```

Output ▶

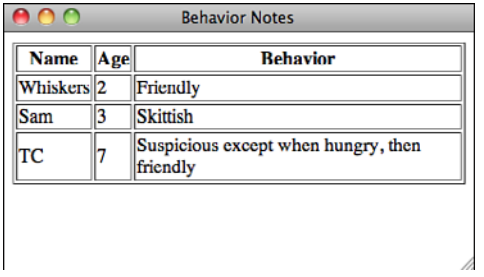


FIGURE 10.15
A table with one long row.

By putting in line breaks, you can wrap that row in a shorter column so that it looks more like the table shown in Figure 10.16. The following shows how the revised code looks for the last row:

Input ▼

```
<tr>
  <td>TC</td>
  <td>7</td>
  <td>Suspicious except<br />
    when hungry, <br />
    then friendly</td>
 </tr>
```

Output ▶

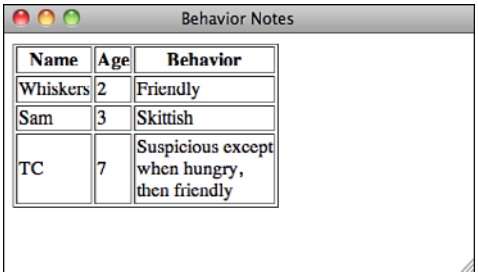


FIGURE 10.16
The long row fixed with `<br/>`.

On the other hand, you might have a table in which a cell is being wrapped and you want all the data on one line. (This can be particularly important for things such as form elements within table cells, where you want the label and the input field to stay together.) In this instance, you can add the `nowrap` attribute to the `<th>` or `<td>` elements, and the browser keeps all the data in that cell on one line. Note that you can always add `<br />` elements to that same cell by hand and get line breaks exactly where you want them.

Let's suppose you have a table where the column headings are wider than the data in the columns. If you want to keep them all online, use `nowrap` as follows:

```
<table width="50%" summary="Best Hitters of All Time">
  <tr>
    <th>Player Name</th>
    <th nowrap="nowrap">Batting Average</th>
    <th nowrap="nowrap">Home Runs</th>
    <th>RBI</th>
  </tr>
  <tr>
    <td>Babe Ruth</td>
    <td>.342</td>
    <td>714</td>
    <td>2217</td>
  </tr>
  <tr>
    <td>Ted Williams</td>
    <td>.344</td>
    <td>521</td>
    <td>1839</td>
  </tr>
</table>
```

Regardless of the width of the table, the Batting Average and the Home Runs column headings will not wrap.

NOTE

The nowrap attribute has been deprecated in HTML 4.01 in favor of using style sheet properties.

To achieve the same results in CSS, use the white-space property. The default value is "normal". To disable word wrapping, use the value "nowrap". Here's how one of the headings in the previous example would be written using CSS:

```
<th style="white-space: normal">Batting Average</th>
```

Be careful when you hard-code table cells with line breaks and nowrap attributes. Remember, your table might be viewed by users with many different screen widths. Try resizing the browser window to make sure your table still looks correct. For the most part, try to let the browser format your table and make minor adjustments only when necessary.

Table and Cell Color

After you have your basic table layout with rows, headings, and data, you can start refining how that table looks. You can refine tables in a couple of ways. One way is to add color to borders and cells.

There are two ways to change the background color of a table, a row, or a cell inside a row. In the pre-CSS world, you would use the bgcolor attribute of the <table>, <tr>, <th>, or <td> elements. Just as in the <body> tag, the value of bgcolor is a color specified as a hexadecimal triplet or one of the 16 color names: Black, White, Green, Maroon, Olive, Navy, Purple, Gray, Red, Yellow, Blue, Teal, Lime, Aqua, Fuchsia, or Silver. In the style sheet world, you use the background-color property or the background property. You can use the style attribute in the <table>, <tr>, <th>, and <td> elements, just as you can in most other elements. Each background color overrides the background color of its enclosing element. For example, a table background overrides the page background, a row background overrides the table's, and any cell colors override all other colors. If you nest tables inside cells, that nested table has the background color of the cell that encloses it.

Also, if you change the color of a cell, don't forget to change the color of the text inside it so that you can still read it.

NOTE

For table cells to show up with background colors, they must not be empty. Simply putting a `<br />` element in empty cells works fine.

Here's an example of changing the background and cell colors in a table. I've created a checkerboard using an HTML table. The table itself is white, with alternating cells in black. The checkers (here, red and black circles) are images. In the source code, I've used both `bgcolor` and the `background-color` property to set background colors for some of the cells. As you'll see in the screenshot, the appearance of both is the same when rendered in the browser:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Checkerboard</title>
</head>
<body>
<table bgcolor="#FFFFFF" width="50%" summary="checkerboard">
  <tr align="center">
    <td bgcolor="#000000" width="33%"><img src="redcircle.png" /></td>
    <td width="33%"><img src="redcircle.png" alt="" /></td>
    <td bgcolor="#000000" width="33%"><img src="redcircle.png" /></td>
  </tr>

  <tr align="center">
    <td><img src="blackcircle.png" alt="" /></td>
    <td style="background-color: #000000"><br /></td>
    <td><img src="blackcircle.png" alt="" /></td>
  </tr>

  <tr align="center">
    <td bgcolor="#000000"><br /></td>
    <td><img src="blackcircle.png" alt="" /></td>
    <td bgcolor="#000000"><br /></td>
  </tr>
</table>
</body>
</html>
```

Figure 10.17 shows the result.

Output ▶

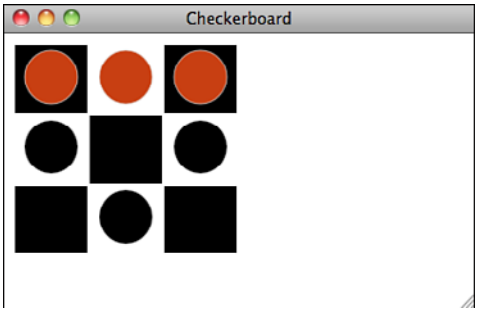


FIGURE 10.17
Table cell colors.

DO	DON'T
DO test your tables with various sizes of browser windows to make sure they look okay.	**DON'T** use tables just to put borders around elements on a page; use CSS.
DO increase the `cellpadding` in your tables to make them more readable.	**DON'T** use tables just to apply a background color to an element; use CSS instead.
	DON'T use tables format nontabular data if you can help it.

Aligning Your Table Content

Another enhancement that you can make to your tables is to adjust the alignment of their content. The `align` attribute aligns content horizontally, whereas the `valign` attribute aligns content vertically, and of course, you can use CSS properties to accomplish the same things, too. The following sections describe how to use these attributes in tables.

Table Alignment

By default, tables are displayed on a line by themselves along the left side of the page, with any text above or below the table. However, you can use the `align` attribute to align tables along the left or right margins and wrap text alongside them the same way you can with images.

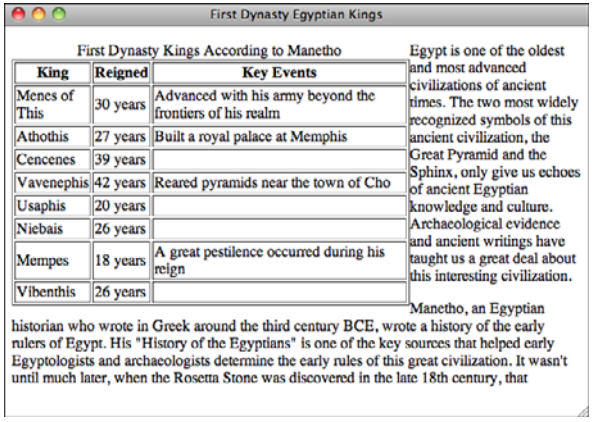
`align="left"` aligns the table along the left margin, and all text following that table is wrapped in the space between that table and the right side of the page. `align="right"` does the same thing, with the table aligned to the right side of the page.

In the example shown in Figure 10.18, a table that spans 70% of the width of the page is aligned to the left with the following code:

```
<table border="1" align="left" width="70%">
```

As you can see from the screenshot, one problem with wrapping text around tables is that HTML has no provision for creating margins that keep the text and the image from jamming right up next to each other. That problem can be addressed by applying a margin to the table using CSS.

FIGURE 10.18
A table with text alongside it.



As with images, you can use the line break element with the `clear` attribute to stop wrapping text alongside a table. Centering tables is slightly more difficult. Most browsers support the `align="center"` attribute in `table` tags.

TIP

The CSS approach to flowing content around a table is to use the `float` property, as described in Lesson 7, "Formatting Text with HTML and CSS." So to align a table to the right and float content around it, you could use a style declaration like this:

```
.sidetable { width: 300xp; float: right; margin: 15px; }
```

That makes the table 300 pixels wide, aligns it to the right, and applies a 15 pixel margin to all four sides. To center a table on the page, you can use a margin value of `auto`, as follows:

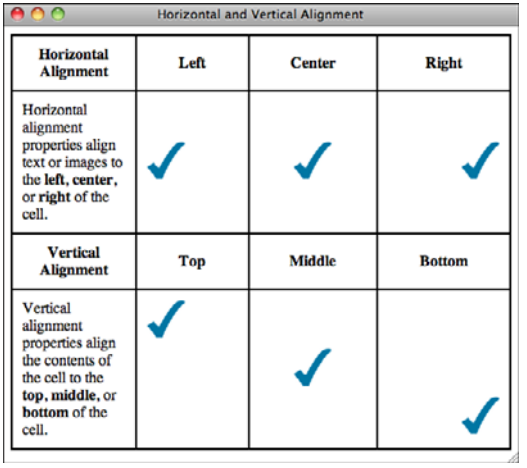
```
.centered { margin-left: auto; margin-right: auto; }
```

Cell Alignment

After you have your rows and cells in place inside your table and the table is properly aligned on the page, you can align the data within each cell for the best effect, based on what your table contains. Several options enable you to align the data within your cells

both horizontally and vertically. Figure 10.19 shows a table (a real HTML one!) of the various alignment options.

FIGURE 10.19
Aligned content within cells.



Horizontal alignment (the `align` attribute) defines whether the data within a cell is aligned with the left cell margin (`left`), the right cell margin (`right`), or centered within the two (`center`). The one place where the `align` attribute wasn't deprecated in XHTML 1.0 is the `<td>` and `<th>` tags. It's perfectly okay to use it within your tables. That said, it is more efficient to align the contents of cells using CSS if you want to apply the alignment to many cells at once.

Vertical alignment (the `valign` attribute) defines the vertical alignment of the data within the cell: flush with the top of the cell (`top`), flush with the bottom of the cell (`bottom`), or vertically centered within the cell (`middle`). Newer browsers also implement `valign="baseline"`, which is similar to `valign="top"` except that it aligns the baseline of the first line of text in each cell. (Depending on the contents of the cell, this might or might not produce a different result than `valign="top"`.)

By default, heading cells are centered both horizontally and vertically, and data cells are centered vertically but aligned flush left.

You can override the defaults for an entire row by adding the `align` or `valign` attributes to the `<tr>` element, as in the following:

```
<tr align="center" valign="top">
```

You can override the row alignment for individual cells by adding `align` to the `<td>` or `<th>` elements:

```
<tr align="center" valign="top">
  <td>14</td>
  <td>16</td>
  <td align=left>No Data</td>
  <td>15</td>
</tr>
```

The following input and output example shows the various cell alignments and how they look (see Figure 10.20). I've added a style sheet that sets the cell heights to 100 pixels to make the vertical alignments easier to see:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Cell Alignments</title>
    <style type="text/css">
        td { height: 100px; }
    </style>
</head>
<body>
<table border="1" cellpadding="8" width="100%">
  <tr>
    <th><br /></th>
    <th>Left</th>
    <th>Centered</th>
    <th>Right</th>
  </tr>

  <tr>
    <th>Top</th>
    <td align="left" valign="top"><img src="star.png" alt="" /></td>
    <td align="center" valign="top"><img src="star.png" alt="" /></td>
    <td align="top" valign="top"><img src="star.png" alt="" /></td>
  </tr>

  <tr>
    <th>Centered</th>
    <td align="left" valign="middle"><img src="star.png" alt="" /></td>
    <td align="center" valign="middle"><img src="star.png" alt="" /></td>
    <td align="right" valign="middle"><img src="star.png" alt="" /></td>
  </tr>
```

```
  <tr>
    <th>Bottom</th>
    <td align="left" valign="bottom"><img src="star.png" alt="" /></td>
    <td align="center" valign="bottom"><img src="star.png" alt="" /></td>
    <td align="right" valign="bottom"><img src="star.png" alt="" /></td>
  </tr>
</table>
</body>
</html>
```

Output ▶

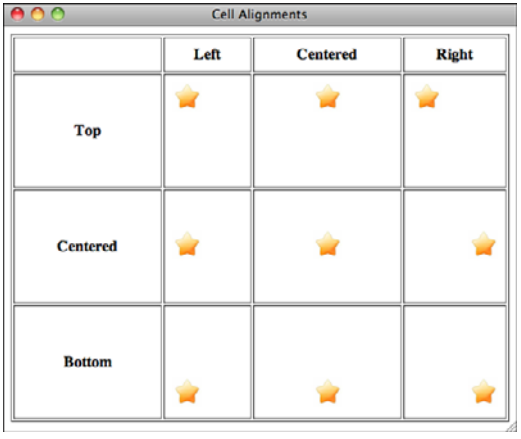


FIGURE 10.20
A matrix of cell alignment settings.

Caption Alignment

The optional `align` attribute of the `<caption>` tag determines the alignment of the caption. Depending on which browser you're using, however, you have different choices for what `align` means.

There are four values for the `align` attribute of the `<caption>` tag, `top`, `bottom`, `left`, and `right`. By default, the caption is placed at the top of the table (`align="top"`). You can use the `align="bottom"` attribute to the caption if you want to put the caption at the bottom of the table, like the following:

```
<table>
<caption align="bottom">Torque Limits for Various Fruits</caption>
```

Similarly, `left` places the caption to the left of the table, and `right` places it to the right.

In Internet Explorer, however, captions are handled slightly differently. The `top` and `bottom` values are treated in the standard fashion, but left and right are different. Rather than placing the caption to the side of the table specified, they align the caption horizontally on the top or bottom of the table, and the placement of the caption is then left to the

nonstandard `valign` attribute. So, in Internet Explorer you could place a caption at the bottom of the table, aligned with the right edge like this:

```
<table>
<caption valign="bottom" align="right">Torque Limits for Various Fruits</cap-
tion>
```

To create the same effect in all current browsers, you can use a combination of HTML and CSS. To place the caption at the bottom right of the table, you would use the `align` attribute and `text-align` property as follows:

```
<caption align="bottom" style="text-align: right">This is a caption</caption>
```

In general, unless you have a very short table, you should leave the caption in its default position—centered at the top of the table. That way your visitors will see the caption first and know what they're about to read, instead of seeing it after they're already done reading the table (at which point they've usually figured out what it's about anyway).

The `align` attribute was removed from HTML5. You should use the standard `align` CSS property instead.

Spanning Multiple Rows or Columns

The tables you've created up to this point all had one value per cell or the occasional empty cell. You also can create cells that span multiple rows or columns within the table. Those spanned cells then can hold headings that have subheadings in the next row or column, or you can create other special effects within the table layout. Figure 10.21 shows a table with spanned columns and rows.

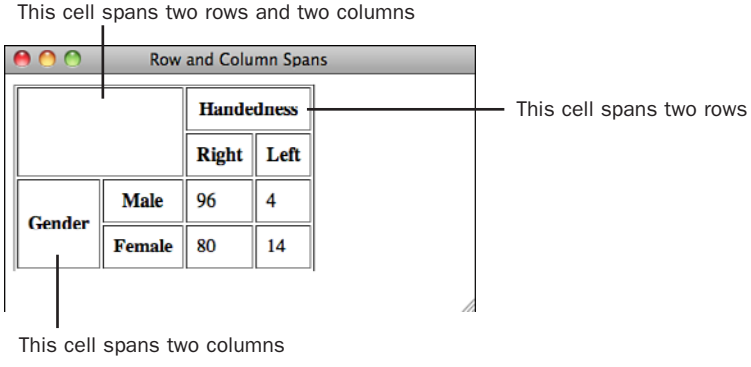


FIGURE 10.21
Using span settings to alter table layout.

To create a cell that spans multiple rows or columns, you add the `rowspan` or `colspan` attribute to the `<th>` or `<td>` elements, along with the number of rows or columns you want the cell to span. The data within that cell then fills the entire width or length of the combined cells, as in the following example:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Row and Column Spans</title>
</head>
<body>
<table border="1" summary="span example">
  <tr>
    <th colspan="2">Gender</th>
  </tr>

  <tr>
    <th>Male</th>
    <th>Female</th>
  </tr>

  <tr>
    <td>15</td>
    <td>23</td>
  </tr>
</table>
</body>
</html>
```

Figure 10.22 shows how this table might appear when displayed.

Output ▶

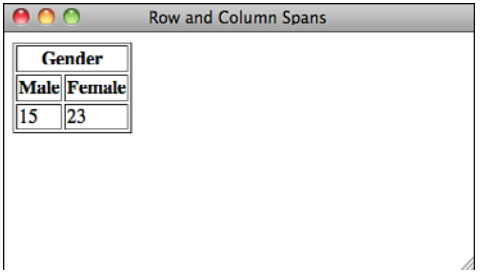


FIGURE 10.22
Using span settings to widen a column.

Note that if a cell spans multiple rows, you don't have to redefine it as empty in the next row or rows. Just ignore it and move to the next cell in the row. The span fills in the spot for you.

Cells always span downward and to the right. To create a cell that spans several columns, you add the `colspan` attribute to the leftmost cell in the span. For cells that span rows, you add `rowspan` to the topmost cell.

The following input and output example shows a cell that spans multiple rows (the cell with the word *Piston* in it). Figure 10.23 shows the result.

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Ring Clearance</title>
</head>
<body>
<table border="1" summary="ring clearance">
  <tr>
    <th colspan="2"> </th>
    <th>Ring<br />
        Clearance</th>
  </tr>

  <tr align="center">
    <th rowspan="2">Piston</th>
    <th>Upper</th>
    <td>3mm</td>
  </tr>

  <tr align="center">
    <th>Lower</th>
    <td>3.2mm</td>
  </tr>
</table>
</body>
</html>
```

Output ▶

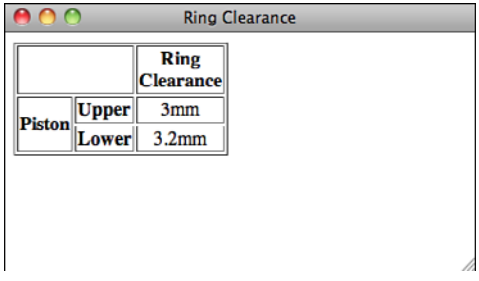


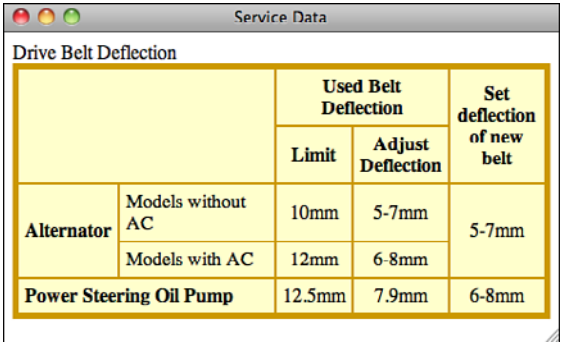
FIGURE 10.23
Cells that span multiple rows and columns.

Task: Exercise 10.2: A Table of Service Specifications

Had enough of tables yet? Let's do another example that takes advantage of everything you've learned here: tables that use colors, headings, normal cells, alignments, and column and row spans. This is a complex table, so we'll go step by step, row by row, to build it.

Figure 10.24 shows the table, which indicates service and adjustment specifications from the service manual for a car.

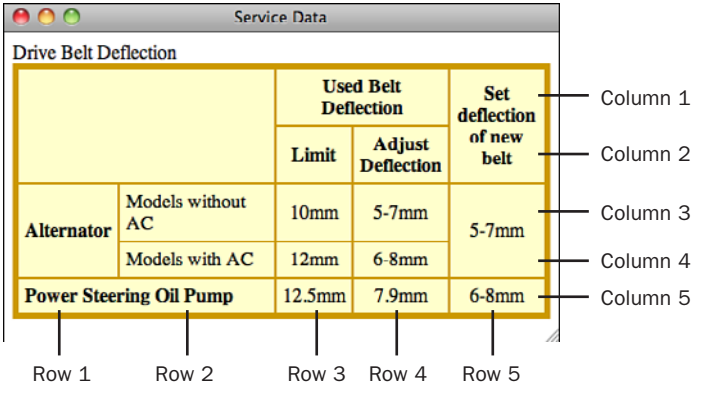
FIGURE 10.24
The complex service specification table.



There are actually five rows and columns in this table. Do you see them? Some of them span columns and rows. Figure 10.25 shows the same table with a grid drawn over it so that you can see where the rows and columns are.

With tables such as this one that use many spans, it's helpful to draw this sort of grid to figure out where the spans are and in which row they belong. Remember, spans start at the topmost row and the leftmost column.

FIGURE 10.25
Five columns, five rows.



Ready? Start with the framework, just as you have for the other tables in this lesson:

```
<!DOCTYPE html>
<html>
<head>
<title>Service Data</title>
</head>
<body>
<table border="1" summary="drive belt deflection">
<caption>Drive Belt Deflection</caption>
</table>
</body>
</html>
```

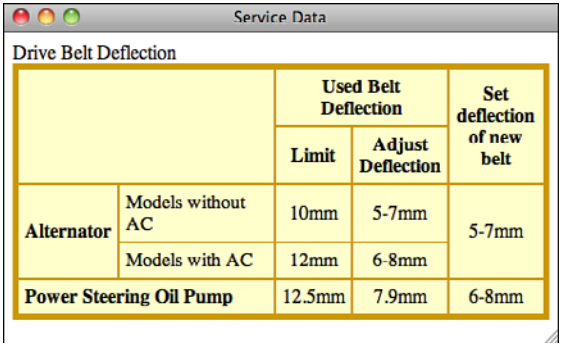
To enhance the appearance of the table, make all the cells light yellow (`#ffffcc`) by using the `background-color` property. The border will be increased in size to 5 pixels, and you'll color it deep gold (`#cc9900`) by using the `border` property. You'll make the rules between cells appear solid by using a `cellspacing` setting of `0` and increase the white space between the cell contents and the borders of the cells by specifying a `cellpadding` setting of `5`. The new table definition now looks like the following:

```
<table summary="drive belt deflection"
style="background-color: #ffffcc; border: 5px solid #cc9900"
  cellspacing="0"cellpadding="5">
```

Now create the first row. With the grid on your picture, you can see that the first cell is empty and spans two rows and two columns (see Figure 10.26). Therefore, the HTML for that cell would be as follows:

```
<tr>
<th rowspan="2" colspan="2"></th>
```

FIGURE 10.26
The first cell.



The second cell in the row is the Used Belt Deflection heading cell, which spans two columns (for the two cells beneath it). The code for that cell is as follows:

```
<th colspan="2">Used Belt Deflection</th>
```

Now that you have two cells that span two columns each, there's only one left in this row. However, this one, like the first one, spans the row beneath it:

```
<th rowspan="2">Set deflection of new belt</th>
</tr>
```

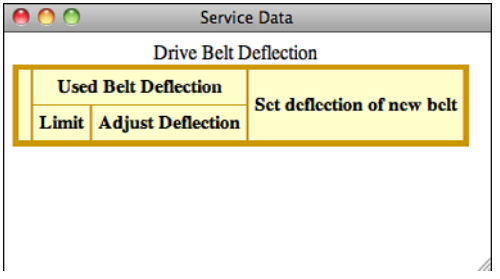
Now go on to the second row. This isn't the one that starts with the Alternator heading. Remember that the first cell in the previous row has a `rowspan` and a `colspan` of two, meaning that it bleeds down to this row and takes up two cells. You don't need to redefine it for this row. You just move on to the next cell in the grid. The first cell in this row is the Limit heading cell, and the second cell is the Adjust Deflection heading cell:

```
<tr>
  <th>Limit</th>
  <th>Adjust Deflection</th>
</tr>
```

What about the last cell? Just like the first cell, the cell in the row above this one had a `rowspan` of `2`, which takes up the space in this row. The only values you need for this row are the ones you already defined.

Are you with me so far? Now is a great time to try this out in your browser to make sure that everything is lining up. It'll look kind of funny because you haven't really put anything on the left side of the table yet, but it's worth a try. Figure 10.27 shows what you've got so far.

FIGURE 10.27
The table so far.



Next row! Check your grid if you need to. Here, the first cell is the heading for Alternator, and it spans this row and the one below it:

```
<tr>
  <th rowspan="2">Alternator</th>
```

Are you getting the hang of this yet?

The next three cells are pretty easy because they don't span anything. Here are their definitions:

```
<td>Models without AC</td>
<td>10mm</td>
<td>5-7mm</td>
```

The last cell in this row is just like the first one:

```
<td rowspan="2">5-7mm</td>
</tr>
```

You're up to row number four. In this one, because of the `rowspans` from the previous row, there are only three cells to define: the cell for Models with AC and the two cells for the numbers:

```
<tr>
  <td>Models with AC</td>
  <td>12mm</td>
  <td>6-8mm</td>
</tr>
```

NOTE

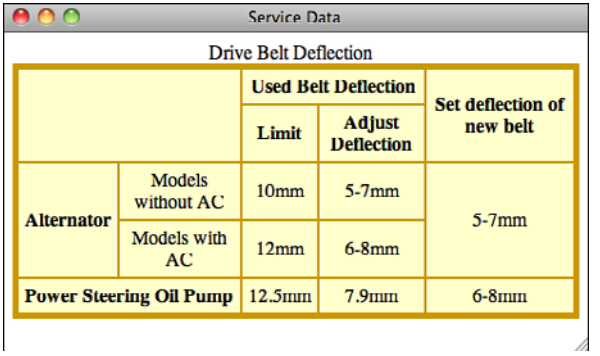
In this table, I've made the Alternator cell a heading cell and the AC cells plain data. This is mostly an aesthetic decision on my part. I could have made all three into headings just as easily.

Now for the final row—this one should be easy. The first cell (Power Steering Oil Pump) spans two columns (the one with Alternator in it and the with/without AC column). The remaining three are just one cell each:

```
<tr>
  <th colspan="2">Power Steering Oil Pump</th>
  <td>12.5mm</td>
  <td>7.9mm</td>
  <td>6-8mm</td>
</tr>
```

That's it. You're done laying out the rows and columns. That was the hard part. The rest is just fine-tuning. Try looking at it again to make sure there are no strange errors (see Figure 10.28).

FIGURE 10.28
The table with the data rows included.



Now that you have all the rows and cells laid out, adjust the alignments within the cells. The numbers should be centered, at least. Because they make up the majority of the table, center the default alignment for each row:

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

The labels along the left side of the table (Alternator, Models with/without AC, and Power Steering Oil Pump) look funny if they're centered, however, so left-align them using the following code:

```
<th rowspan="2" style="text-align: left">Alternator</th>
<td style="text-align: left">Models without AC</td>
<td style="text-align: left">Models with AC</td>

<th colspan="2" style="text-align: left">Power Steering Oil Pump</th>
```

I've put some line breaks in the longer headings so that the columns are a little narrower. Because the text in the headings is pretty short to start with, I don't have to worry too much about the table looking funny if it gets too narrow. Here are the lines I modified:

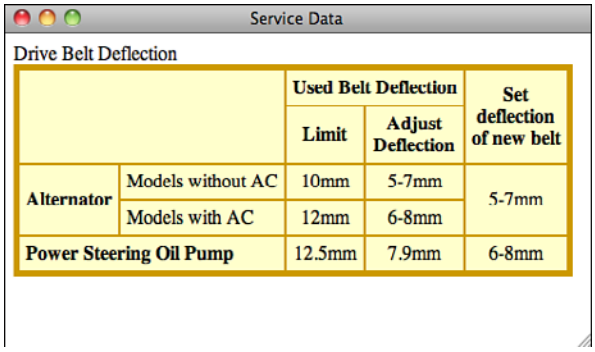
```
<th rowspan="2">Set<br />deflection<br />of new belt</th>
<th>Adjust<br />Deflection</th>
```

For one final step, you'll align the caption to the left side of the table:

```
<caption style="text-align: left">Drive Belt Deflection</caption>
```

Voilá—the final table, with everything properly laid out and aligned! Figure 10.29 shows the final result.

FIGURE 10.29
The final Drive Belt Deflection table.



TIP

If you got lost at any time, the best thing you can do is pull out your handy text editor and try it yourself, following along tag by tag. After you've done it a couple of times, it becomes easier.

Here's the full text for the table example:

```
<!DOCTYPE html>
<html>
<head>
<title>Service Data</title>
<style type="text/css">
        td, th { border: 1px solid #cc9900; }
        table { background-color: #ffffcc; border: 4px solid #cc9900;  }
</style>
</head>
<body>
<table summary="drive belt deflection"
cellspacing="0"
cellpadding="5">
<caption style="text-align: left">Drive Belt Deflection</caption>
<tr>
  <th rowspan="2" colspan="2"></th>
  <th colspan="2">Used Belt Deflection</th>
  <th rowspan="2">Set<br />deflection<br />of new belt</th>
</tr>
<tr>
  <th>Limit</th>
  <th>Adjust<br />Deflection</th>
</tr>
<tr style="text-align: center">
  <th rowspan="2" style="text-align: left">Alternator</th>
  <td style="text-align: left">Models without AC</td>
```

```
    <td>10mm</td>
    <td>5-7mm</td>
    <td rowspan="2">5-7mm</td>
  </tr>
  <tr style="text-align: center">
    <td style="text-align: left">Models with AC</td>
    <td>12mm</td>
    <td>6-8mm</td>
  </tr>
  <tr style="text-align: center">
    <th colspan="2" style="text-align: left">Power Steering Oil Pump</th>
    <td>12.5mm</td>
    <td>7.9mm</td>
    <td>6-8mm</td>
  </tr>
  </table>
  </body>
  </html>
```

NOTE

Under normal circumstances, avoid the use of the `style` attribute and instead use a style sheet for the page and apply classes where necessary to style your table. Using the `style` attribute is the least efficient way to apply styles to a page, but it makes the example more readable.

More Advanced Table Enhancements

Tables are laid out row by row, but HTML also provides some elements that enable you to group cells into columns and modify their properties. There are also elements that enable you to group the rows in tables to manage them collectively as well.

Grouping and Aligning Columns

Sometimes it's helpful to apply styles to the columns in your tables, rather than applying them to individual cells or to rows. To do so, you need to define the columns in your table with the `<colgroup>` and `<col>` elements.

The `<colgroup>...</colgroup>` element is used to enclose one or more columns in a group. The closing `</colgroup>` tag is optional in HTML, but it's required by XHTML. This element has two attributes:

- `span` defines the number of columns in the column group. Its value must be an integer greater than `0`. If `span` isn't defined, the `<colgroup>` element defaults to a

column group that contains one column. If the <colgroup> element contains one or more <col> elements (described later), however, the span attribute is ignored.

- width specifies the width of each column in the column group. Widths can be defined in pixels, percentages, and relative values. You also can specify a special width value of "0*" (zero followed by an asterisk). This value specifies that the width of each column in the group should be the minimum amount necessary to hold the contents of each cell in the column. If you specify the "0*" value, however, browsers will be unable to render the table incrementally (meaning that all the markup for the table will have to be downloaded before the browser can start displaying it).

Suppose that you have a table that measures 450 pixels in width and contains six columns. You want each of the six columns to be 75 pixels wide. The code looks something like the following:


```
<table border="1" width="450">
<colgroup span="6" width="75">
</colgroup>
```

Now you want to change the columns. Using the same 450-pixel-wide table, you make the first two columns 25 pixels wide, and the last four columns 100 pixels wide. This requires two <colgroup> elements, as follows:

```
<table border="1" width="450">
<colgroup span="2" width="25">
</colgroup>
<colgroup span="4" width="100">
</colgroup>
```

What if you don't want all the columns in a column group to be the same width or have the same appearance? That's where the <col> element comes into play. Whereas <colgroup> defines the structure of table columns, <col> defines their attributes. To use this element, begin the column definition with a <col> tag. The end tag is forbidden in this case. Instead, use the XHTML 1.0 construct for tags with no closing tag and write the tag as <col />.

Going back to your 450-pixel-wide table, you now want to make the two columns in the first column group 75 pixels wide. In the second column group, you have columns of 50, 75, 75, and 100 pixels, respectively. Here's how you format the second column group with the <col> tag:

```
<table border="1" width="450">
  <colgroup span="2" width="75" />
  </colgroup>
  <colgroup>
    <col span="1" width="50" />
    <col span="2" width="75" />
    <col span="1" width="100" />
  </colgroup>
```

Now apply this to some *real* code. The following example shows a table that displays science and mathematics class schedules. Start by defining a table that has a one-pixel-wide border and spans 100% of the browser window width.

Next, you define the column groups in the table. You want the first column group to display the names of the classes. The second column group consists of two columns that display the room number for the class, as well as the time that the class is held. The first column group consists of one column of cells that spans 20% of the entire width of the table. The contents of the cell are aligned vertically toward the top and centered horizontally. The second column group consists of two columns, each spanning 40% of the width of the table. Their contents are vertically aligned to the top of the cells. To further illustrate how `colgroup` works, I use the `style` attribute and `background-color` property to set each of the column groups to have different background colors.

Finally, you enter the table data the same way that you normally do. Here's what the complete code looks like for the class schedule, and the results are shown in Figure 10.30:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Grouping Columns</title>
</head>
<body>
<table border="1" width="100%" summary="Grouping Columns">
  <caption><b>Science and Mathematic Class Schedules</b></caption>

  <colgroup width="20%" align="center" valign="top"
  style="background-color: #fcf"></colgroup>

  <colgroup span="2" width="40%" valign="top"
  style="background-color: #ccf"></colgroup>
```

```
  <tr>
    <th>Class</th>
    <th>Room</th>
    <th>Time</th>
  </tr>

  <tr>
    <td>Biology</td>
    <td>Science Wing, Room 102</td>
    <td>8:00 AM to 9:45 AM</td>
  </tr>

  <tr>
    <td>Science</td>
    <td>Science Wing, Room 110</td>
    <td>9:50 AM to 11:30 AM</td>
  </tr>

  <tr>
    <td>Physics</td>
    <td>Science Wing, Room 107</td>
    <td>1:00 PM to 2:45 PM</td>
  </tr>

  <tr>
    <td>Geometry</td>
    <td>Mathematics Wing, Room 236</td>
    <td>8:00 AM to 9:45 AM</td>
  </tr>

  <tr>
    <td>Algebra</td>
    <td>Mathematics Wing, Room 239</td>
    <td>9:50 AM to 11:30 AM</td>
  </tr>

  <tr>
    <td>Trigonometry</td>
    <td>Mathematics Wing, Room 245</td>
    <td>1:00 PM to 2:45 PM</td>
  </tr>
</table>
</body>
</html>
```

Output ▶

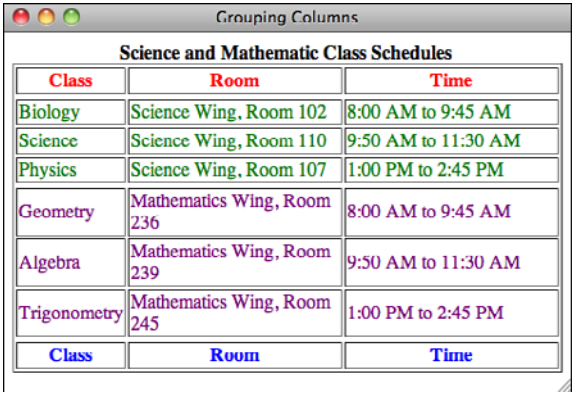


FIGURE 10.30
The class schedule with formatted column groups.

Grouping and Aligning Rows

Now that you know how to group and format columns, let's turn to the rows. You can group the rows of a table into three sections: table heading, table footer, and table body. You can modify CSS properties to emphasize the table heading and table footer and give the body of the table a different appearance.

The table header, footer, and body sections are defined by the `<thead>`, `<tfoot>`, and `<tbody>` elements, respectively. Each of these elements must contain the same number of columns.

The `<thead>...</thead>` element defines the heading of the table, which should contain information about the columns in the body of the table. Typically, this is the same type of information that you've been placing within header cells so far in the lesson. The starting `<thead>` tag is always required when you want to include a head section in your table, as is the closing `</thead>` tag under XHTML 1.0.

The head of the table appears right after the `<table>` element or after `<colgroup>` elements, as the following example shows, and must include at least one row group defined by the `<tr>` element. I'm including `style` attributes in the row grouping tags to illustrate how they are used. The table is formatted as follows:

Input ▼

```
<table border="1" width="100%" summary="Science and Mathematic Class Schedules">
  <caption><b>Science and Mathematic Class Schedules</b></caption>
  <colgroup width="20%" align="center" valign="top">
  <colgroup span="2" width="40%" valign="top">
  <thead style="color: red">
  <tr>
```

```
    <th>Class</th>
    <th>Room</th>
    <th>Time</th>
  </tr>
</thead>
```

The `<tfoot>...</tfoot>` element defines the footer of the table. The starting `<tfoot>` tag is always required when defining the footer of a table. The closing `<tfoot>` tag was optional in HTML 4.01, but it's required for XHTML 1.0 compliance. The footer of the table appears immediately after the table heading if one is present or after the `<table>` element if a table heading isn't present. It must contain at least one row group, defined by the `<tr>` element. A good example of information that you could place in a table footer is a row that totals columns of numbers in a table.

You must define the footer of the table before the table body because the browser has to render the footer before it receives all the data in the table body. For the purposes of this example, we'll include the same information in the table head and the table footer. The code looks like this:

Input ▼

```
<tfoot style="color: blue">
  <tr>
  <th>Class</th>
  <th>Room</th>
  <th>Time</th>
  </tr>
</tfoot>
```

After you define the heading and footer for the table, you define the rows in the table body. A table can contain more than one body element, and each body can contain one or more rows of data. This might not seem to make sense, but using multiple body sections enables you to divide up your table into logical sections. I show you one example of why this is rather cool in a little bit.

The `<tbody>...</tbody>` element defines a body section within your table. The `<tbody>` start tag is required if at least one of the following is true:

- The table contains head or foot sections.
- The table contains more than one table body.

The following example contains two table bodies, each consisting of three rows of three cells each. The body appears after the table footer, as follows:

Input ▼

```
<tbody style="color: yellow">
  <tr>
    <td>Biology</td>
    <td>Science Wing, Room 102</td>
    <td>8:00 AM to 9:45 AM</td>
  </tr>
  <tr>
    <td>Science</td>
    <td>Science Wing, Room 110</td>
    <td>9:50 AM to 11:30 AM</td>
  </tr>
  <tr>
    <td>Physics</td>
    <td>Science Wing, Room 107</td>
    <td>1:00 PM to 2:45 PM</td>
  </tr>
</tbody>
<tbody style="color: grey">
  <tr>
    <td>Geometry</td>
    <td>Mathematics Wing, Room 236</td>
    <td>8:00 AM to 9:45 AM</td>
  </tr>
  <tr>
    <td>Algebra</td>
    <td>Mathematics Wing, Room 239</td>
    <td>9:50 AM to 11:30 AM</td>
  </tr>
  <tr>
    <td>Trigonometry</td>
    <td>Mathematics Wing, Room 245</td>
    <td>1:00 PM to 2:45 PM</td>
  </tr>
</tbody>
</table>
```

Put all the preceding together and you get a table that looks like that shown in Figure 10.31.

Output ▶

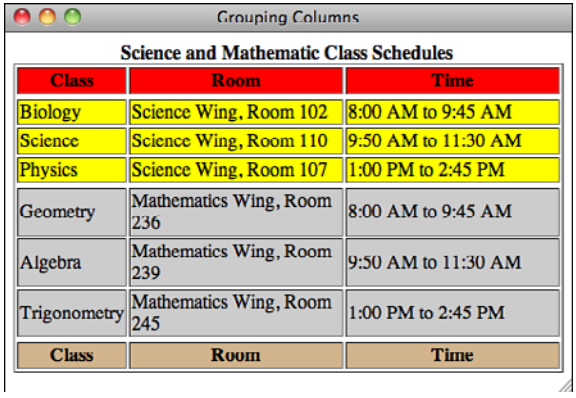


FIGURE 10.31
The class schedule with a head, two bodies, and a foot.

Other Table Elements and Attributes

Table 10.1 presents some of the additional elements and attributes that pertain to tables.

TABLE 10.1 Other Table Elements and Attributes

Attribute	Applied to Element	Use
`char`	See "Use" column	Specifies a character to be used as an axis to align the contents of a cell. For example, you can use it to align a decimal point in numeric values. Can be applied to `colgroup`, `col`, `tbody`, `thead`, `tfoot`, `tr`, `td`, and `th` elements. Removed from HTML5.
`charoff`	See "Use" column	Specifies the amount of offset applied to the first occurrence of the alignment character that is specified in the `char` attribute. Applies to `colgroup`, `col`, `tbody`, `thead`, `tfoot`, `tr`, `td`, and `th` elements. Removed from HTML5.
`summary`	`<table>`	Provides a more detailed description of the contents of the table and is primarily used with nonvisual browsers.

How Tables Are Used

In this lesson, I explained the usage of tables in publishing tabular data. That was the original purpose for HTML tables. However, in 1996, Netscape 2.0 introduced the option of turning off table borders, and this, along with other limitations in HTML, changed the way tables were used.

Before style sheets were invented and implemented in most browsers, there was only one way to lay out elements on a page other than straight down the middle: tables. These days, developers use CSS to lay out pages, but before CSS support in browsers became really solid, tables were the key page layout tool that most web developers used.

Even now, there are some cases where using tables to lay out pages make sense. The web browsers in some mobile devices do not support CSS, so if you want to lay out your pages with columns, you must use tables. Similarly, if you are creating a web page that will be sent out as part of an email message, tables should be used. Some email clients do not support CSS, and so for more advanced layouts you're required to use tables.

Summary

In this lesson, you learned quite a lot about tables. They enable you to arrange your information in rows and columns so that your visitors can get to the information they need quickly.

While working with tables, you learned about headings and data, captions, defining rows and cells, aligning information within cells, and creating cells that span multiple rows or columns. With these features, you can create tables for most purposes.

As you're constructing tables, it's helpful to keep the following steps in mind:

- Sketch your table, indicating where the rows and columns fall. Mark which cells span multiple rows and columns.
- Start with a basic framework and lay out the rows, headings, and data row by row and cell by cell in HTML. Include row and column spans as necessary. Test frequently in a browser to make sure that it's all working correctly.
- Modify the alignment in the rows to reflect the alignment of the majority of the cells.
- Modify the alignment for individual cells.
- Adjust line breaks, if necessary.
- Make other refinements such as cell spacing, padding, or color.
- Test your table in multiple browsers. Different browsers might have different approaches to laying out your table or might be more accepting of errors in your HTML code.

Table 10.2 presents a quick summary of the HTML elements that you learned about in this lesson, and which remain current in HTML 4.01.

TABLE 10.2 Current HTML 4.01 Table Elements

Tag	Use
`<table>...</table>`	Indicates a table.
`<caption>...</caption>`	Creates a caption for the table (optional).
`<colgroup>...</colgroup>`	Encloses one or more columns in a group.
`<col>`	Used to define the attributes of a column in a table.
`<thead>...</thead>`	Creates a row group that defines the heading of the table. A table can contain only one heading.
`<tfoot>...</tfoot>`	Creates a row group that defines the footer of the table. A table can contain only one footer. Must be specified before the body of the table is rendered.
`<tbody>...</tbody>`	Defines one or more row groups to include in the body of the table. Tables can contain more than one body section.
`<tr>...</tr>`	Defines a table row, which can contain heading and data cells.
`<th>...</th>`	Defines a table cell that contains a heading. Heading cells are usually indicated by boldface and centered both horizontally and vertically within the cell.
`<td>...</td>`	Defines a table cell containing data. Table cells are in a regular font and are left-aligned and vertically centered within the cell.

Because several of the table attributes apply to more than one of the preceding elements, I'm listing them separately. Table 10.3 presents a quick summary of the HTML attributes you learned about in this lesson that remain current in HTML 4.01.

TABLE 10.3 Current HTML 4.01 Table Attributes

Attribute	Applied to Element	Use
`align`	`<tr>`	Possible values are `left`, `center`, and `right`, which indicate the horizontal alignment of the cells within that row (overriding the default alignment of heading and table cells).
	`<th>` or `<td>`	Overrides both the row's alignment and any default cell alignment. Possible values are `left`, `center`, and `right`.
	`<thead>`, `<tbody>`, `<tfoot>`	Used to set alignment of the contents in table head, body, or foot cells. Possible values are `left`, `center`, and `right`.

TABLE 10.3 Continued

Attribute	Applied to Element	Use
	`<col>`	Used to set alignment of all cells in a column. Possible values are `left`, `center`, and `right`.
	`<colgroup>`	Used to set alignment of all cells in a column group. Possible values are `left`, `center`, and `right`.
	`<table>`	Deprecated in HTML 4.01. Possible values are `left`, `center`, and `right`. `align="center"` isn't supported in HTML 3.2 and older browsers. Determines the alignment of the table and indicates that text following the table will be wrapped alongside it.
	`<caption>`	Deprecated in HTML 4.01. Indicates which side of the table the caption will be placed. The possible values for most browsers are `top` and `bottom`. HTML 4.01 browsers also support `left` and `right`. In Internet Explorer, the possible values are `left`, `right`, and `center`, and they indicate the horizontal alignment of the caption.
`bgcolor`	`All`	(HTML 3.2, deprecated in HTML 4.01.) Changes the background color of that table element. Cell colors override row colors, which override table colors. The value can be a hexadecimal color number or a color name.
`border`	`<table>`	Indicates whether the table will be drawn with a border. The default is no border. If `border` has a value, it's the width of the shaded border around the table.
`bordercolor`	`<table>`	(Internet Explorer extension.) Can be used with any of the table elements to change the color of the border around that elements. The value can be a hexadecimal color number or a color name.

TABLE 10.3 Continued

Attribute	Applied to Element	Use
bordercolorlight	<table>	(Internet Explorer extension.) Same as bordercolor, except it affects only the light component of a 3D-look border.
bordercolordark	<table>	(Internet Explorer extension.) Same as bordercolor, except it affects only the dark component of a 3D-look border.
cellspacing	<table>	Defines the amount of space between the cells in the table.
cellpadding	<table>	Defines the amount of space between the edges of the cell and its contents.
char		Specifies a character to be used as an axis to align the contents of a cell (for example, a decimal point in numeric values). Can be applied to colgroup, col, tbody, thead, tfoot, tr, td, and th elements.
charoff		Specifies the amount of offset to be applied to the first occurrence of the alignment character specified by the char attribute. Applies to the same elements previously listed in char.
frame	<table>	Defines which sides of the frame that surrounds a table are visible. Possible values are void, above, below, hsides, lhs, rhs, vsides, box, and border.
height	<th> or <td>	Deprecated in HTML 4.01. Indicates the height of the cell in pixel or percentage values.
nowrap	<th> or <td>	Deprecated in HTML 4.01. Prevents the browser from wrapping the contents of the cell.
rules	<table>	Defines which rules (division lines) appear between cells in a table. Possible values are none, groups, rows, cols, and all.
width	<table>	Indicates the width of the table, in exact pixel values or as a percentage of page width (for example, 50%).

TABLE 10.3 Continued

Attribute	Applied to Element	Use
span	<colgroup>	Defines the number of columns in a column group. Must be an integer greater than 0.
	<col>	Defines the number of columns that a cell spans. Must be an integer greater than 0.
width	<colgroup>	Defines the width of all cells in a column group.
	<col>	Defines the width of all cells in one column.
colspan	<th> or <td>	Indicates the number of cells to the right of this one that this cell will span.
rowspan	<th> or <td>	Indicates the number of cells below this one that this cell will span.
valign	<tr>	Indicates the vertical alignment of the cells within that row (overriding the defaults). Possible values are top, middle, and bottom.
	<th> or <td>	Overrides both the row's vertical alignment and the default cell alignment. Possible values are top, middle, bottom, and baseline.
	<thead>, <tfoot>, <tbody>	Defines vertical alignment of cells in the table head, table foot, or table body.
	<colgroup>	Defines the vertical alignment of all cells in a column group.
	<col>	Defines the vertical alignment of all cells in a single column.
width	<th> or <td>	Deprecated in HTML 4.01. Indicates width of the cell in exact pixel values or as a percentage of table width (for example, 50%).

Workshop

This lesson covered one of the more complex subjects in HTML: tables. Before you move on to the next lesson, you should work through the following questions and exercises to make sure that you've got a good grasp of how tables work.

Q&A

Q **Tables are a real hassle to lay out, especially when you get into row and column spans. That last example was awful.**

A You're right. Tables are a tremendous pain to lay out by hand like this. However, if you're using writing editors and tools to generate HTML code, having the table defined like this makes more sense because you can just write out each row in turn programmatically.

Q **Can you nest tables, putting a table inside a single table cell?**

A Sure! As I mentioned earlier, you can put any HTML code you want inside a table cell, and that includes other tables.

Q **Is there a way to specify a beveled border like the default table borders using CSS?**

A CSS actually provides three different beveled border styles: `inset`, `outset`, and `ridge`. You should experiment with them and use the one that looks the best to you.

Quiz

1. What are the basic parts of a table, and which tags identify them?
2. Which attribute is the most common attribute of the table tag, and what does it do?
3. What attributes define the amount of space between the edges of the cells and their content, and the amount of space between cells?
4. Which attributes are used to create cells that span more than one column or row?
5. Which elements are used to define the head, body, and foot of a table?

Quiz Answers

1. The basic parts of a table (the `<table>` tag) are the border (defined with the `border` attribute), caption (defined with the `<caption>` tag), header cells (`<th>`), data cells (`<td>`), and table rows (`<tr>`).
2. The `border` attribute is the most common attribute for the table tag. It specifies whether border lines are displayed around the table and how wide the borders should be.
3. `cellpadding` defines the amount of space between the edges of the cell and their contents. `cellspacing` defines the amount of space between the cells.
4. The `rowspan` attribute creates a cell that spans multiple rows. The `colspan` attribute creates a cell that spans multiple columns.
5. `<thead>`, `<tbody>`, and `<tfoot>` define the head, body, and foot of a table.

Exercises

1. Here's a brainteaser for you: Create a simple nested table (a table within a table) that contains three rows and four columns. Inside the cell that appears at the second column in the second row, create a second table that contains two rows and two columns.
2. One tricky aspect of working with the HTML for tables is accounting for cells with no data. Create a table that includes empty cells and verify that when you've done so, all the rows and columns line up as you originally anticipated.

LESSON 11
Designing Forms

Up to this point, you've learned almost everything you need to know to create functional, attractive, and somewhat interactive web pages. If you think about it, however, the pages you've created thus far have a one-way information flow. This lesson is about creating HTML forms to collect information from people visiting your website. Forms enable you to gather just about any kind of information for immediate processing by a server-side script or for later analysis using other applications

This lesson covers the following topics, which enable you to create any type of form possible with HTML:

- Discovering how HTML forms interact with server-side scripts to provide interactivity
- Creating simple forms to get the hang of it
- Learning all the types of form controls you can use to create radio buttons, check boxes, and more
- Using more advanced form controls to amaze your friends and co-workers
- Planning forms so that your data matches any server-side scripts you use

Understanding Form and Function

Right off the bat, you need to understand a few things about forms. First, a form is part of a web page that you create using HTML elements. Each form contains a `form` element that contains special controls, such as buttons, text fields, check boxes, Submit buttons, and menus. These controls make up the user interface for the form (that is, the pieces of the form users see on the web page). When people fill out forms, they're interacting with these elements. In addition, you can use many other HTML elements within forms to create labels, provide additional information, add structure, and so on. These elements aren't part of the form itself, but they can enhance your form's look and improve its usability.

When someone fills out an HTML form, he enters information or makes choices using the form controls. When the user submits the form, the browser collects all the data from the form and sends it to the URL specified as the form's action. It's up to the program residing at that URL to process the form input and create a response for the user.

It's important that you understand the implications of this final step. The data is what you want, after all! This is the reason you've chosen to create a form in the first place. When a user clicks the Submit button, the process ceases to be one of pure HTML and becomes reliant on applications that reside on the web server. In other words, for your form to work, you must already have a program on the server that will store or manipulate the data in some manner.

In some cases, forms aren't necessarily submitted to programs. Using JavaScript, you can take action based on form input. For example, you can open a new window when a user clicks a form button. You can also submit forms via email, which is okay for testing but isn't reliable enough for real applications.

▼ Task: Exercise 11.1: Creating a Simple Form That Accepts a Name and a Password

Okay, let's get right to it and create a simple form that illustrates the concepts just presented. It's a web page that prompts the user to enter a name and a password to continue.

Start by opening your favorite HTML editor and creating a web page template. Enter the standard HTML header information, include the `body` element, and then close the `body` and `html` elements to form a template from which to work. If you already have a template similar to this, just load it into your HTML editor:

```
<!DOCTYPE html>=
<html>
```

```
<head>
<title>Page Title</title>
</head>
<body>


</body>
</html>
```

Next, add your `title` so that people will understand the purpose of the web page:

```
<title>Please Log In</title>
```

Within the `body` of the web page, add a `form` element. I've added both the opening and closing tags, with an empty line between them, so that I don't forget to close the `form` when I'm finished:

```
<form action="/form-processing-script" method="post">

</form>
```

Before continuing, you need to know more about the `form` element and the attributes you see within the opening tag. Obviously, `form` begins the element and indicates that you're creating an HTML form. The `action` attribute specifies the URL to the server-side script (including the filename) that will process the form when it's submitted. It's important that the script with the name you've entered is present on your web server at the location the URL specifies. In this example, I use the full URL for the script, but you can just as easily use a relative URL if it makes more sense.

CAUTION

Before going live with forms, contact your web hosting provider and ask whether you can use the hosting provider's scripts or add your own. You must also determine the URL that points to the directory on the server that contains the scripts. Some hosting providers rigidly control scripts for security purposes and won't allow you to create or add scripts to the server. If that's the case, and you really need to implement forms on your web pages, consider searching for a new hosting provider.

The next attribute is `method`, which can accept one of two possible values: `post` or `get`. These values define how form data is submitted to your web server. The `post` method includes the form data in the body of the form and sends it to the web server. The `get` method appends the data to the URL specified in the `action` attribute and most often is

used in searches. I chose the `post` method here because I don't want to send the user's password back in plain sight as part of the URL. Now add some form controls and information to make it easy for a visitor to understand how to fill out the form. Within the `form` element, begin by adding a helpful description of the data to be entered by the user, and then add a `text` form control. This prompts the user to enter her name in a text-entry field. Don't worry about positioning just yet because you'll put all the form controls into a table later:

```
<form action="/form-processing-script" method=
post>
  <label for="username">Username</label> <input type="text" name="username" />
</form>
```

Next, add another bit of helpful text and a password control:

```
<form action="/form-processing-script" method="post">
  <label for="username">Username</label> <input type="text" name="username" />

  <label for="password">Password</label> <input type="password" name="password"
/>
</form>
```

Notice that both these form controls are created using the `input` element. The `type` attribute defines which type of control will be created. In this case, you have a text control and a password control. Each type of control has a distinct appearance, accepts a different type of user input, and is suitable for different purposes. Each control is also assigned a name that distinguishes it and its data from the other form controls.

The labels for the form fields are specified using the `<label>` tag. Each label is attached to the form field it is associated with through the `for` attribute, which should match the `name` or `id` attribute of the form tag with which it is associated. The `<label>` element doesn't provide any formatting by default, but you can make it appear however you want using CSS.

Finally, add a Submit button so that the user can send the information she entered into the form. Here's the form so far:

Input

```
<form action="/form-processing-script " method="post">
    <div>
        <label for="username">Username</label>
        <input type="text" name="username" />
    </div>
```

```
    <div>
        <label for="password">Password</label>
        <input type="password" name="password" />
    </div>

    <div>
        <input type="submit" class="submit" value="Log In" />
    </div>
</form>
```

The Submit button is another type of `input` field. The `value` attribute is used as the label for the Submit button. If you leave it out, the default label will be displayed by the browser.

TIP

When you're naming form controls and labeling buttons, strive for clarity and meaning. If a form is frustrating or hard to figure out, visitors will leave your site for greener pastures!

Figure 11.1 contains a screenshot of the form with all the form elements in place.

Output ▶

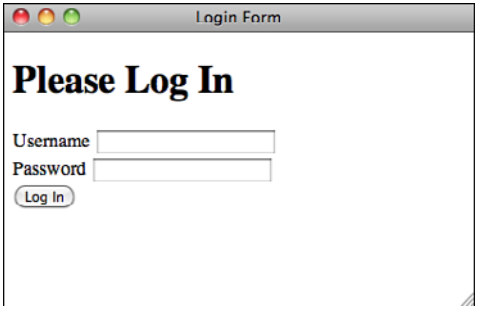


FIGURE 11.1
The form with all the input elements in place.

At this point, you've created the form and it's ready to rumble. However, if you load it into your web browser, you'll see that it doesn't look all that appealing. I can vastly improve the appearance using *Cascading Style Sheets (CSS)*. Here's the code for the full page, including the style sheet:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login Form</title>
    <style type="text/css">
        div {
```

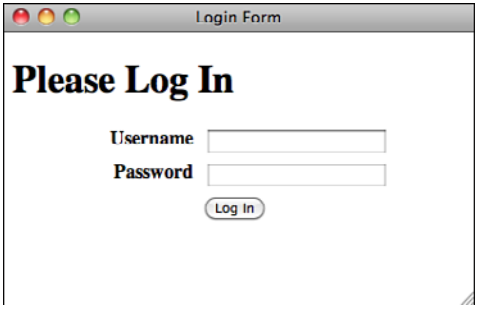
```
            margin-bottom: 5px;
        }

        label {
            display: block;
            float: left;
            width: 150px;
            text-align: right;
            font-weight: bold;
            margin-right: 10px;
        }

        input.submit {
            margin-left: 160px;
        }
    </style>
</head>
<body>

<h1>Please Log In</h1>

<form action="/form-processing-script" method="post">
    <div>
        <label for="username">Username</label>
        <input type="text" name="username" />
    </div>

    <div>
        <label for="password">Password</label>
        <input type="password" name="password" />
    </div>

    <div>
        <input type="submit" class="submit" value="Log In" />
    </div>
</form>

</body>
</html>
```

At one time, it was rare to see forms that were laid out without the use of tables, but tables are no longer necessary thanks to CSS. Let's look at the style sheet for the form.

First, I added 5 pixels of margin to the bottom of the `<div>` elements to space out the form elements a bit. Then, I used CSS to align the form fields vertically and right-align the labels. You can only apply widths to block-level elements, so I set the `display` property on the labels to `block`. Then I used `float: left` and a width of 150 pixels to get the form fields to move to the right of the labels. Setting the `text-align` property to

right for the labels moves them to the right side of the 150-pixel box I put them in. Then I just added a 10-pixel margin to create some space between the labels and the form fields and bolded the label text. To get the Submit button, which has no label, to line up with the other form fields, I added a 160-pixel right margin. That's 150 pixels for the label and 10 pixels for the margin I added to the labels. That took a little work, but I think the final page shown in Figure 11.2 looks good.

FIGURE 11.2
A simple form.



To complete the exercise, let's test the form to see whether the form produces the data we expect. Here's what the data that's sent to the server looks like:

```
username=someone&password=somepassword
```

It's pretty measly, but you can see that the names assigned to each field are tied to the values entered in those fields. You can then use a program to use this data to process the user's request.

Using the <form> Tag

To accept input from a user, you must wrap all your input fields inside a <form> tag. The purpose of the <form> tag is to indicate where and how the user's input should be sent. First, let's look at how the <form> tag affects page layout. Forms are block-level elements. That means when you start a form, a new line is inserted (unless you apply the display: inline CSS property to the form tag).

NOTE

Form controls must appear inside another block level element inside the <form> element to be considered valid. A , <p>, or <table> will all do the trick, as will other block level elements, such as the <fieldset> tag, which I talk about a bit further on.

Take a look at the following code fragment:

Input ▼

```
<p>Please enter your username <form><input /> and password
<input /></form> to log in.</p>
```

You might think that your entire form would appear on a single line based on the preceding markup. As shown in Figure 11.3, the opening and closing `<form>` tags act like opening and closing paragraph tags.

Output ▶

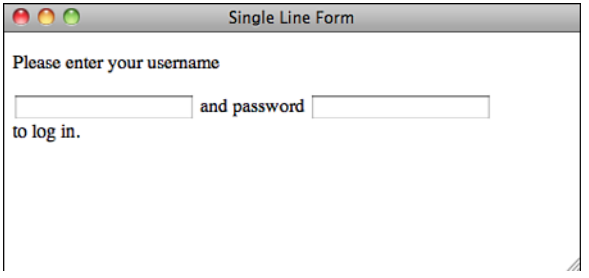


FIGURE 11.3
A line break inserted by an opening *<form>* tag.

The two most commonly used attributes of the `<form>` tag are `action` and `method`. Both of these attributes are optional. The following example shows how the `<form>` tag is typically used:

```
<form action="someaction" method="get or post">
content, form controls, and other HTML elements
</form>
```

`action` specifies the URL to which the form is submitted. Again, remember that for the form to be submitted successfully, the script must be in the exact location you specify and must work properly.

If you leave out the `action` attribute, the form is submitted to the current URL. In other words, if the form appears on the page http://www.example.com/form.html and you leave off the `action` attribute, the form will be submitted to that URL by default. This probably doesn't seem very useful, but it is if your form is generated by a program instead of residing in an HTML file. In that case, the form is submitted back to that program for processing. One advantage of doing so is that if you move the program on the server, you don't have to edit the HTML to point the form at the new location.

Although most forms send their data to scripts, you also can make the action link to another web page or a `mailto` link. The latter is formed as follows:

```
<form action="mailto:somebody@isp.com" method="post">
```

This attaches the form data set to an email, which then is sent to the email address listed in the `action` attribute.

TIP

To test your forms, I recommend using the `get` method and leaving out the `action` attribute of the form tag. When you submit the form, the values you entered will appear in the URL for the page so that you can inspect them and make sure that the results are what you expected.

The `method` attribute supports two values: `get` or `post`. The method indicates how the form data should be packaged in the request that's sent back to the server. The `get` method appends the form data to the URL in the request. The form data is separated from the URL in the request by a question mark and is referred to as the *query string*. If I have a text input field named `searchstring` and enter `Orangutans` in the field, the resulting would look like the following:

```
http://www.example.com/search?searchstring=Orangutans
```

The `method` attribute is not required; if you leave it out, the `get` method will be used. The other method is `post`. Instead of appending the form data to the URL and sending the combined URL-data string to the server, `post` sends the form data to the location specified by the `action` attribute in the body of the request.

DO	DON'T
DO use the `POST` method when data on the server will be changed in any way.	**DON'T** use the `GET` method if you do not want the form parameters to be visible in a URL.
DO use the `GET` method if the form just requests data like search forms, for example.	**DON'T** use the `GET` method if the form is used to delete information.
DO use the `GET` method if you want to bookmark the results of the form submission.	

The general rule when it comes to choosing between `post` and `get` is that if the form will change any data on the server, you should use `post`. If the form is used to retrieve information, using `get` is fine. For example, suppose that you're writing a message board program. The registration form for new users and the form used to publish messages should use the `post` method. If you have a form that enables the user to show all the posts entered on a certain date, it could use the `get` method.

One other less frequently used attribute of the `<form>` tag is `enctype`. It defines how form data is encoded when it's sent to the server. The default is `application/x-www-form-urlencoded`. The only time you ever need to use `enctype` is when your form includes a file upload field (which I discuss a bit later). In that case, you need to specify that the `enctype` is `multipart/form-data`. Otherwise, it's fine to leave it out.

That about does it for the `<form>` tag, but you've really only just begun. The `<form>` tag alone is just a container for the input fields that are used to gather data from users. It just indicates where the data should go and how it should be packaged. To actually gather information, you're going to need items called form controls.

Using the `<label>` Tag

Whenever you enter text that describes a form field, you should use the `<label>` tag, and use the `for` attribute to tie it to the control it labels. To create a label, begin with the opening `label` tag and then enter the `for` attribute. The value for this attribute, when present, must match the `id` or `name` attribute for the control it labels. Next, enter text that will serve as the label and then close the element with the end `label` tag, as in the following:

Input ▼

```
<label for="control4">Who is your favorite NFL Quarterback?</label>
<input type="text" name="favqb" id="control4" />
```

Figure 11.4 shows this text control with a label assigned to it.

Output ▶

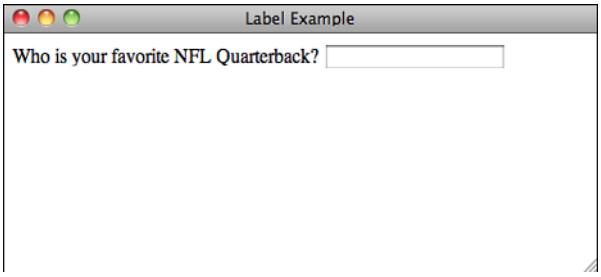


FIGURE 11.4
You can assign labels to any form control. Note that they're displayed with the control.

If you include your form control within the `label` element, as shown in the following code, you can omit the `for` attribute:

```
<label>User name <input type="text" name="username" /></label>
```

The `<label>` tag doesn't cause any visible changes to the page, but you can style it using CSS, as you saw in the example login form earlier. One common styling approach people use is to apply a special style to the labels of required fields. For example, you may declare a style rule like this:

```
label.required { font-weight: bold }
```

You can then set the `class` for the labels for all the required fields in your form to "required," and the labels for those fields will appear in boldface.

Creating Form Controls with the <input> Tag

Now it's time to learn how to create the data entry fields form. The `<input>` tag enables you to create many different types of form controls.

Form controls are special HTML tags used in a form that enable you to gather information from visitors to your web page. The information is packaged into a request sent to the URL in the form's `action` attribute.

The `input` element consists of an opening tag with attributes, no other content, and no closing tag:

```
<input attributes />
```

The key point here is using the right attributes to create the form control you need. The most important of these is `type`, which specifies what kind of form control to display. For all controls, except Submit and Reset buttons, the `name` attribute is required. It

associates a name with the data entered in that field when the data is sent to the server. The rest of this section describes the different types of controls you can create using the `input` element.

Creating Text Controls

Text controls enable you to gather information from a user in small quantities. This control type creates a single-line text input field in which users can type information, such as their name or a search term.

To create a text input field, create an `input` element and choose `text` as the value for the `type` attribute. Make sure to name your control so that the server script can process the value:

Input ▼

```
<label for="petname">Enter your pet's name</label>
<input type="text" name="petname" />
```

Figure 11.5 shows this text control, which tells the user what to type in.

Output ▶

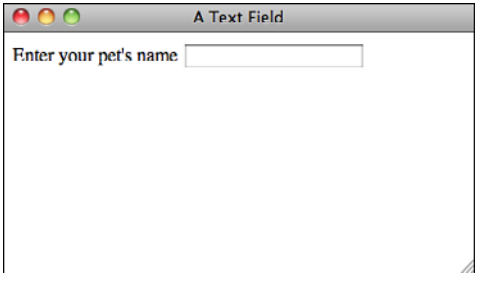


FIGURE 11.5
A text entry field.

You can modify the appearance of text controls using the `size` attribute. Entering a number sets the width of the text control in characters:

```
<input type="text" name="petname" size="15" />
```

To limit the number of characters a user can enter, add the `maxlength` attribute to the text control. This doesn't affect the appearance of the field; it just prevents the user from entering more characters than specified by this attribute. If users attempt to enter more text, their web browsers will stop accepting input for that particular control:

```
<input type="text" name="petname" size="15" maxlength="15" />
```

To display text in the text control before the user enters any information, use the `value` attribute. If the user is updating data that already exists, you can specify the current or default value using `value`, or you can prompt the user with a value:

```
<input type="text" name="petname" size="15" maxlength="15" value="Enter Pet
Name" />
```

In this case, `Enter Pet Name` appears in the field when the form is rendered in the web browser. It remains there until the user modifies it.

CAUTION

When you're using the `value` attribute, using a value that's larger than the size of the text control can confuse the user because the text will appear to be cut off. Try to use only enough information to make your point. Ensure that any `value` is less than or equal to the number of characters you specified in `size`.

Creating Password Controls

The `password` and `text` field types are identical in every way except that the data entered in a password field is masked so that someone looking over the shoulder of the person entering information can't see the value that was typed into the field.

TIP

You don't have to limit your use of the `password` control to just passwords. You can use it for any sensitive material that you feel needs to be hidden when the user enters it into the form.

To create a password control, create an `input` element with the `type` set to `password`. To limit the size of the password control and the maximum number of characters a user can enter, you can use the `size` and `maxlength` attributes just as you would in a `text` control. Here's an example:

Input ▼

```
<label for="userpassword">Enter your password</label> <input type="password"
name="userpassword"
  size="8" maxlength="8" />
```

Figure 11.6 shows a password control.

Output ▶

FIGURE 11.6
A password form field.

CAUTION

When data entered in a `password` field is sent to the server, it is not encrypted in any way. Therefore, this is not a secure means of transmitting sensitive information. Although the users can't read what they are typing, the `password` control provides no other security measures.

Creating Submit Buttons

Submit buttons are used to indicate that the user is finished filling out the form. Setting the `type` attribute of the form to `submit` places a Submit button on the page with the default label determined by the browser, usually `Submit Query`. To change the button text, use the `value` attribute and enter your own label, as follows:

```
<input type="submit" value="Send Form Data" />
```

NOTE

Your forms can contain more than one Submit button.

If you include a `name` attribute for a Submit button, the `value` that you assign to the field is sent to the server if the user clicks on that Submit button. This enables you to take different actions based on which Submit button the user clicks, if you have more than one. For example, you could create two Submit buttons, both with the `name` attribute set to `"action"`. The first might have a value of `"edit"` and the second a value of `"delete"`. In your script, you could test the value associated with that field to determine what the user wanted to do when he submitted the form.

Creating Reset Buttons

Reset buttons set all the form controls to their default values. These are the values included in the `value` attributes of each field in the form (or in the case of selectable fields, the values that are preselected). As with the Submit button, you can change the label of a Reset button to one of your own choosing by using the `value` attribute, like this:

```
<input type="reset" value="Clear Form" />
```

CAUTION

Reset buttons can be a source of some confusion for users. Unless you have a really good reason to include them on your forms, you should probably just avoid using them. If your form is large and the user clicks the Reset button when he means to click the Submit button, he isn't going to be pleased with having to go back and reenter all of his data.

Creating Check Box Controls

Check boxes are fields that can be set to two states: on and off (see Figure 11.7). To create a check box, set the `input` tag's type attribute to `checkbox`. The `name` attribute is also required, as shown in the following example:

Input ▼

```
<label>Check to receive SPAM email <input type="checkbox" name="spam" /></label>
```

Output ▶

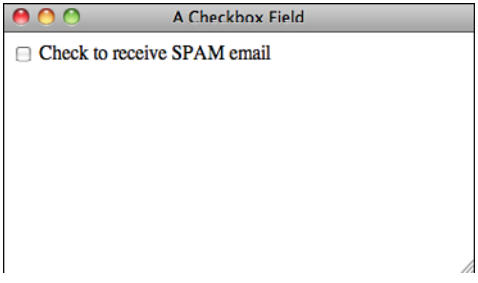


FIGURE 11.7
A check box field.

To display the check box as checked, include the `checked` attribute, as follows:

```
<input type="checkbox" name="year" checked="checked" />
```

You can group check boxes together and assign them the same control name. This allows multiple values associated with the same name to be chosen:

```
<p>Check all symptoms that you are experiencing:</p>
<label><input type="checkbox" name="symptoms" value="nausea" /> Nausea</label>
<label><input type="checkbox" name="symptoms" value="lightheadedness" />
Light-headedness</label>
<label><input type="checkbox" name="symptoms" value="fever" /> Fever</label>
<label><input type="checkbox" name="symptoms" value="headache" />
Headache</label>
```

When this form is submitted to a script for processing, each check box that's checked returns a value associated with the name of the check box. If a check box isn't checked, neither the field name nor value will be returned to the server—it's as if the field didn't exist at all.

You may have noticed that when I applied labels to these check box elements, I put the `input` tags inside the `label` tags. There's a specific reason for doing so. When you associate a label with a check box (or with a radio button, as you'll see in the next section), the browser enables you to check the box by clicking the label as well as by clicking the button. That can make things a bit easier on your user.

In the examples thus far, I have tied labels to fields by putting the field name in the `for` attribute of the label, but that doesn't work for check boxes because multiple form fields have the same name, and the browser would not figure out which check box the label applies to. Instead I put the `input` tag inside the `label` tag. I could achieve the same result by assigning a unique `id` to each check box and associating them with the labels that way, but nesting the tags is easier.

Creating Radio Buttons

Radio buttons, which generally appear in groups, are designed so that when one button in the group is selected, the other buttons in the group are automatically unselected. They enable you to provide users with a list of options from which only one option can be selected. To create a radio button, set the `type` attribute of an `<input>` tag to `radio`. To create a radio button group, set the `name` attributes of all the fields in the group to the same value, as shown in Figure 11.8. To create a radio button group with three options, the following code is used:

Input ▼

```
<p>Select a color:</p>
<label style="display: block"><input type="radio" name="color" value="red" />
Red</label>
<label style="display: block"><input type="radio" name="color" value="blue" />
Blue</label>
```

```
<label style="display: block"><input type="radio" name="color" value="green" />
Green</label>
```

Output ▶

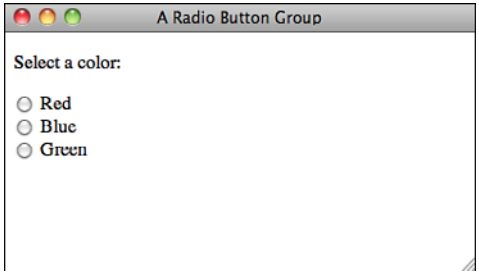


FIGURE 11.8
A group of radio buttons.

I've used the same `<label>` technique here that I did in the check box example. Placing the radio buttons inside the labels makes the labels clickable as well as the radio buttons themselves. I've changed the `display` property for the labels to `block` so that each radio button appears on a different line. Ordinarily I'd apply that style using a style sheet; I used the `style` attributes to include the styles within the example.

As with check boxes, if you want a radio button to be selected by default when the form is displayed, use the `checked` attribute. One point of confusion is that even though browsers prevent users from having more than one member of a radio button group selected at once, they don't prevent you from setting more than one member of a group as checked by default. You should avoid doing so yourself.

Using Images as Submit Buttons

Using `image` as the `type` of `input` control enables you to use an image as a Submit button:

Input ▼

```
<input type="image" src="submit.gif" name="submitformbtn" />
```

Figure 11.9 shows a custom button created with an image.

Output ▶

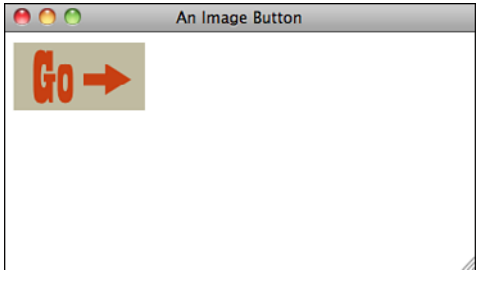


FIGURE 11.9
The image input type.

When the user clicks an image field, the x and y coordinates of the point where the user clicked are submitted to the server. The data is submitted as `name.x = x coord` and `name.y = y coord`, where `name` is the name assigned to the control. Using the preceding code, the result might look like the following:

```
submitoformbtn.x=150&submitformbtn.y=200
```

You can omit the name if you choose. If you do so, the coordinates returned would just be `x =` and `y =`. Form controls with the type `image` support all the attributes of the `<img>` tag. You can also use the same CSS properties you would use with `<img>` tags to modify the appearance and spacing of the button. To refresh your memory on the attributes supported by the `<img>` tag, go back to Lesson 9, "Adding Images, Color, and Backgrounds."

Creating Generic Buttons

In addition to creating Submit, Reset, and Image buttons, you also can create buttons that generate events within the browser that can be tied to client-side scripts. To create such a button, set the `type` attribute to `button`. Figure 11.10 shows a button that calls a function when it is pressed. Use the following code to create a button:

Input ▼

```
<input type="button" name="verify" value="verify" onclick="verifydata()" />
```

Output ▶

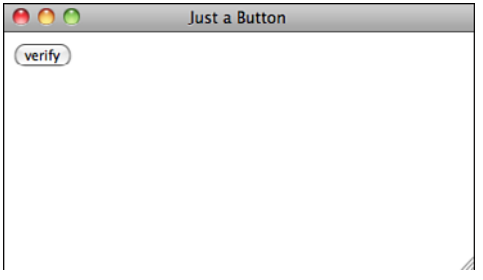


FIGURE 11.10
A button element on a web page.

This example creates a button that runs a function called `verifydata` when it's clicked. You provide the label that appears on the button with the `value` attribute of `Verify Data`.

Unlike Submit buttons, regular buttons don't submit the form when they're clicked. I explain the `onclick` attribute when you get to Lesson 14, "Introducing JavaScript."

Hidden Form Fields

Hidden form fields are used when you want to embed data in a page that shouldn't be seen or modified by the user. The name and value pair associated with a hidden form field will be submitted along with the rest of the contents of the form when the form is submitted. To create such a field, set the field's type to `hidden` and be sure to include both the `name` and `value` attributes in your `<input>` tag. Here's an example:

```
<input type="hidden" name="id" value="1402" />
```

Hidden form fields are generally used when data identifying the user needs to be included in a form. For example, suppose you've created a form that allows a user to edit the name and address associated with her bank account. Because the user can change her name and address, the data she submits can't be used to look up her account after she submits the form, plus there might be multiple accounts associated with one name and address. You can include the account number as a hidden field on the form so that the program on the server knows which account to update when the form is submitted.

CAUTION

It's important to understand that when it comes to hidden form fields, *hidden* means "won't clutter up the page" rather than "won't be discoverable by the user." Anyone can use the View Source feature in the browser to look at the values in hidden form fields, and if you use the `GET` method, those values will appear in the URL when the form is submitted, too. Don't think of hidden fields as a security feature but rather as a convenient way to embed extra data in the form that you know the script that processes the form input will need to use.

The File Upload Control

The file control enables a user to upload a file when he submits the form. As you can see in the following code, the `type` for the input element is set to `file`:

Input ▼

```
<label>Please select a file for upload <input type="file" name="fileupload"
/></label>
```

Figure 11.11 shows a file upload control.

Output ▶

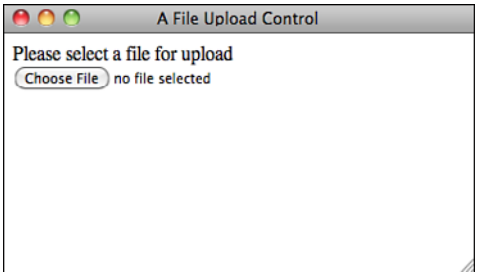


FIGURE 11.11
The file upload control.

If you want to use a file upload field on your form, you have to do a lot of behind-the-scenes work to get everything working. For one thing, the program specified in the `action` attribute of your form must accept the file being uploaded. Second, you have to use the `post` method for the form. Third, you must set the `enctype` attribute of the `<form>` tag to `multipart/form-data`. Ordinarily, the default behavior is fine, but you must change the `enctype` in this particular case.

Let's look at a simple form that supports file uploads:

```
<form action="/upload" enctype="multipart/form-data" method="post">
<input type="file" name="new_file" />
<input type="submit" />
</form>
```

After you've created a form for uploading a file, you need a program that can process the file submission. Creating such a program is beyond the scope of this book, but all popular web programming environments support file uploads.

Using Other Form Controls

In addition to form controls you can create using the `input` element, there are three that are elements in and of themselves.

Using the `button` Element

A button you create using the `button` element is similar to the buttons you create with the `input` element, except that content included between the opening and closing button tags appears on the button.

You can create three different types of buttons: Submit, Reset, and Custom. The `<button>` tag is used to create buttons. As with other form fields, you can use the `name` attribute to specify the name of the parameter sent to the server, and the `value` attribute

to indicate which value is sent to the server when the button is clicked. Unlike buttons created with the <input> tag, the button's label is specified by the content within the <button> tag, as shown in this code:

Input ▼

```
<button type="submit"><b><i>Submit button</i></b></button>
<button type="custom"><img src="recycle.gif"></button>
```

The button element is shown in a browser in Figure 11.12.

Output ▶

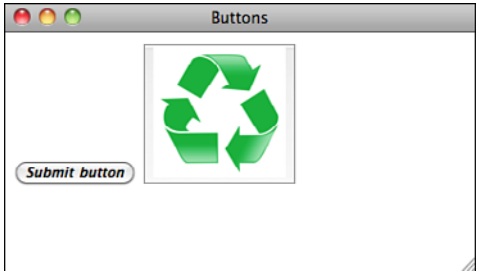


FIGURE 11.12
The *button* element provides a more flexible way to create form buttons.

With the <button> tag, white space counts. If you include whitespace between the opening or closing <button> tags and the content inside the tag, it might make your button look a bit odd. The best bet is to just leave out that whitespace.

Creating Large Text-Entry Fields with `textarea`

The `textarea` element creates a large text entry field where people can enter as much information as they like. To create a `textarea`, use the <textarea> tag. To set the size of the field, use the `rows` and `cols` attributes. These attributes specify the height and width of the text area in characters. A text area with `cols` set to `5` and `rows` set to `40` creates a field that's 5 lines of text high and 40 characters wide. If you leave out the `rows` and `cols` attributes, the browser default will be used. This can vary, so you should make sure to include those attributes to maintain the form's appearance across browsers. The closing `textarea` tag is required and any text you place inside the `textarea` tag is displayed inside the field as the default value:

Input ▼

```
<label for="question4" style="display: block">Please comment on our customer
service.</label>
<textarea name="question4" rows="10" cols="60">Enter your answer here
</textarea>
```

Figure 11.13 shows a `textarea` element in action.

Output ▶

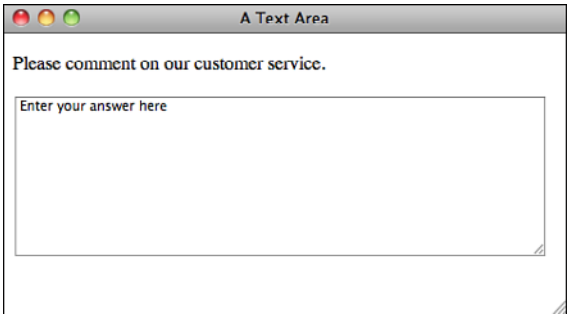


FIGURE 11.13
Use *`textarea`* to create large text-entry areas.

TIP

You can also change the size of a `textarea` with the `height` and `width` CSS properties. You can alter the font in the text area using the CSS font properties, too.

Creating Menus with `select` and `option`

The `select` element creates a menu that can be configured to enable users to select one or more options from a pull-down menu or a scrollable menu that shows several options at once. The `<select>` tag defines how the menu will be displayed and the name of the parameter associated with the field. The `<option>` tag is used to add selections to the menu. The default appearance of select lists is to display a pull-down list that enables the user to select one of the options. Here's an example of how one is created:

Input ▼

```
<label for="location">Please pick a travel destination</label>
<select name="location">
  <option>Indiana</option>
  <option>Fuji</option>
  <option>Timbuktu</option>
  <option>Alaska</option>
</select>
```

As you can see in the code, the field name is assigned using the `name` attribute of the `<select>` tag. The field created using that code appears in Figure 11.14.

Output ▶

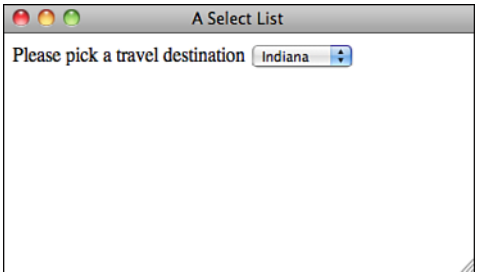


FIGURE 11.14
You can use *select* form controls to create pull-down menus.

To create a scrollable list of items, just include the `size` attribute in the opening `select` tag, like this:

Input ▼

```
<select name="location" size="3">
```

Figure 11.15 shows the same `select` element as Figure 11.14, except that the `size` attribute is set to `3`. Setting the size to `3` indicates that the browser should display three options simultaneously.

Output ▶

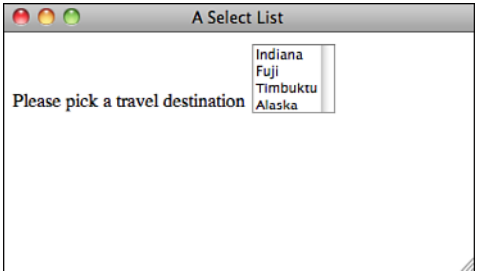


FIGURE 11.15
You also can create scrollable lists using the *select* element.

To see the fourth item, the user would have to use the scrollbar built in to the select list. By default, the value inside the `<option>` tag specifies both what is displayed in the form and what's sent back to the server. To send a value other than the display value to the server, use the `value` attribute. The following code, for example, causes `bw499` to be submitted to the server as the value associated with the `Courses` field instead of `Basket Weaving 499`:

```
<select name="courses">
  <option value="p101">Programming 101</option>
  <option value="e312">Ecomomics 312</option>
  <option value="pe221">Physical Education 221</option>
  <option value="bw499">Basket Weaving 499</option>
</select>
```

To select an option by default, include the `selected` attribute in an `option` element, as follows:

```
<select name="courses">
  <option value="p101">Programming 101</option>
  <option value="e312">Ecomomics 312</option>
  <option value="pe221" selected="selected">Physical Education 221</option>
  <option value="bw499">Basket Weaving 499</option>
</select>
```

Thus far, you've created menus from which a user can select only one choice. To enable users to select more than one option, use the `multiple` attribute:

```
<select name="courses" multiple="multiple">
```

NOTE

A user can choose multiple options by Shift-clicking for Windows, or Ctrl-clicking or Command-clicking for Macintosh.

There are some usability issues associated with select lists. When you think about it, select lists that enable users to choose one option are basically the equivalent of radio button groups, and select lists that allow multiple selections are the same as check box groups. It's up to you to decide which tag to use in a given circumstance. If you need to present the user with a lot of options, select lists are generally the proper choice. A select list with a list of states is a lot more concise and usable than a group of 50 radio buttons. By the same token, if there are four options, radio buttons probably make more sense. The same rules basically hold with check box groups versus multiple select lists.

The other usability issue with select lists is specific to multiple select lists. The bottom line is that they're hard to use. Most users don't know how to select more than one item, and if the list is long enough, as they move through the list they'll have problems keeping track of the items they already selected when they scroll through to select new ones. Sometimes there's no way around using a multiple select list, but you should be careful about it.

▼ Task: Exercise 11.2: Using Several Types of Form Controls

Form controls often come in bunches. Although there are plenty of forms out there that consist of a text input field and a Submit button (like search forms), a lot of the time forms consist of many fields. For example, many websites require that you register to see restricted content, download demo programs, or participate in an online community. In this example, we'll look at a perhaps slightly atypical registration form for a website. ▼

The purpose of this exercise is to show you how to create forms that incorporate a number of different types of form controls. In this case, the form will include a text field, a radio button group, a select list, a check box group, a file upload field, and a text area. The form, rendered in a browser, appears in Figure 11.16.

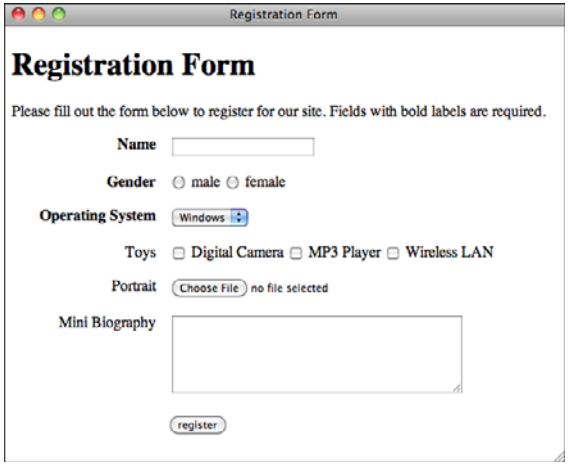


FIGURE 11.16
A registration form for a website.

Let's look at the components used to build the form. The styles for the form are included in the page header. Here's the style sheet:

```
<style type="text/css" media="screen">
    form div {
        margin-bottom: 1em;
    }

    div.submit input {
        margin-left: 165px;
    }

    label.field {
        display: block;
        float: left;
        margin-right: 15px;
        width: 150px;
        text-align: right;
    }

    label.required {
        font-weight: bold;
    }
</style>
```

Looking at the style sheet, you should get some idea of how the form will be laid out. Each field will be in its own `<div>` and I've added a margin to the bottom of each of them. Just as I did in the login form example earlier, I've added a left margin for the Submit button so that it lines up with the other fields. Most of the styling has to do with the labels.

In this form, I am using labels in two ways—first to create a left column of labels for the form, and second to add clickable labels to the radio buttons and check boxes. To distinguish between them, I'm using a class called `field`, which I apply to the field-level labels. I've also got a class called `required` that will be used with labels on required fields.

Now that you've seen the styles, let's look at the body of the page. After some introductory text, we open the form like this:

```
<form action="/register" method="post"
enctype="multipart/form-data">
```

Because this form contains a file upload field, we have to use the `post` method and the `multipart/form-data` `enctype` in the `<form>` tag. The `action` attribute points to a CGI script that lives on my server. Next, we start adding form inputs. Here's the `name` input:

```
        <div>
                <label class="required field" for="name">Name</label>
                <input name="name" />
        </div>
```

All the inputs will follow this basic pattern. The input and its label are nested within a `<div>`. In this case, the label has the classes `field` and `required`. The only attribute included in the input tag is the field name because the default values for the rest of the attributes are fine. Next is the gender field, which uses two radio buttons:

```
        <div>
                <label class="required field">Gender</label>
                <label><input type="radio" name="gender" value="male" />
male</label>
                <label><input type="radio" name="gender" value="female" />
female</label>
        </div>
```

As you can see, the radio button group includes two controls (both with the same name, establishing the group). Because we didn't include line breaks between the two fields, they appear side by side in the form. Here's an instance where I used the `<label>` tag two different ways. In the first case, I used it to label the field as a whole, and then I used

individual labels for each button. The individual labels allow you to select the radio buttons by clicking their labels. As you can see, I used the approach of putting the `<input>` tags inside the `<label>` tags to associate them.

The next field is a select list that enables the user to indicate which operating system he runs on his computer:

```
<div>
    <label class="required field">Operating System</label>

    <select name="os">
        <option value="windows">Windows</option>
        <option value="macos">Mac OS</option>
        <option value="linux">Linux</option>
        <option value="other">Other ...</option>
    </select>
</div>
```

This select list is a single-line, single-select field with four options. Instead of using the display values as the values that will be sent back to the server, we opt to set them specifically using the `value` attribute of the `<option>` tag. The next form field is a check box group:

```
<div>
    <label class="field">Toys</label>
    <label><input type="checkbox" name="toy" value="digicam" /> Digital
Camera</label>
    <label><input type="checkbox" name="toy" value="mp3" /> MP3
Player</label>
    <label><input type="checkbox" name="toy" value="wlan" /> Wireless
LAN</label>
</div>
```

As you can see, we use labels for each of the individual check boxes here, too. The next field is a file upload field:

```
<div>
        <label class="field">Portrait</label>
        <input type="file" name="portrait" />
</div>
```

The last input field on the form is a text area intended for the user's bio.

```
<div>
        <label class="field">Mini Biography</label>
        <textarea name="bio" rows="6" cols="40"></textarea>
</div>
```

After the text area, there's just the Submit button for the form. After that, it's all closing tags for the `<form>` tag, `<body>` tag, and the `<html>` tag. The full source code for the page follows, along with a screenshot of the form, as shown earlier in Figure 11.16.

Input

```
<!DOCTYPE html>
<html>
<head>
<title>Registration Form</title>
<style type="text/css" media="screen">
        form div {
                margin-bottom: 1em;
        }

        div.submit input {
                margin-left: 165px;
        }

        label.field {
                display: block;
                float: left;
                margin-right: 15px;
                width: 150px;
                text-align: right;
        }

        label.required {
                font-weight: bold;
        }
</style>
</head>
<body>
<h1>Registration Form</h1>

<p>Please fill out the form below to register for our site. Fields
with bold labels are required.</p>

<form action="/register" method="post" enctype="multipart/form-data">

        <div>
                <label class="required field" for="name">Name</label>
                <input name="name" />
        </div>

        <div>
                <label class="required field">Gender</label>
                <label><input type="radio" name="gender" value="male" />
```

```
male</label>
                <label><input type="radio" name="gender" value="female" />
female</label>
        </div>

        <div>
                <label class="required field">Operating System</label>

                <select name="os">
                        <option value="windows">Windows</option>
                        <option value="macos">Mac OS</option>
                        <option value="linux">Linux</option>
                        <option value="other">Other ...</option>
                </select>
        </div>

        <div>
                <label class="field">Toys</label>
                <label><input type="checkbox" name="toy" value="digicam" />
Digital Camera</label>
                <label><input type="checkbox" name="toy" value="mp3" /> MP3
Player</label>
                <label><input type="checkbox" name="toy" value="wlan" />
Wireless LAN</label>
        </div>

        <div>
                <label class="field">Portrait</label>
                <input type="file" name="portrait" />
        </div>

        <div>
                <label class="field">Mini Biography</label>
                <textarea name="bio" rows="6" cols="40"></textarea>
        </div>

        <div class="submit">
                <input type="submit" value="register" />
        </div>
</form>
</body>
</html>
```

Grouping Controls with `fieldset` and `legend`

The `fieldset` element organizes form controls into groupings that appear in the web browser. The `legend` element displays a caption for the `fieldset`. To create a `fieldset` element, start with the opening `fieldset` tag, followed by the `legend` element.

Next, enter your form controls and finish things off with the closing `fieldset` tag:

Input ▼

```
<fieldset>
  <legend>Oatmeal Varieties</legend>
  <label><input type="radio" name="applecinnamon" /> Apple Cinnamon</label>
  <label><input type="radio" name="nuttycrunch" /> Nutty Crunch
  </label>
  <label><input type="radio" name="brownsugar" /> Brown Sugar</label>
</fieldset>
```

Figure 11.17 shows the result.

Output ▶

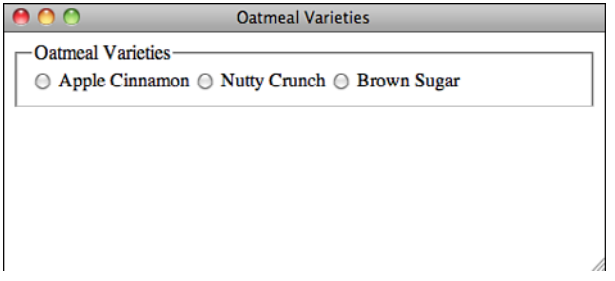


FIGURE 11.17
The *fieldset* and *legend* elements enable you to organize your forms.

The presentation of the `fieldset` in Figure 11.17 is the default, but you can use CSS to style `fieldset` elements in any way that you like. A `fieldset` is a standard block level element, so you can turn off the borders using the style `border: none` and use them as you would `<div>` tags to group inputs in your forms.

One thing to watch out for with the `<legend>` tag is that it's a little less flexible than the `<label>` tag in terms of how you're allowed to style it. It's also not handled consistently between browsers. If you want to apply a caption to a set of form fields, use `<legend>` but be aware that complex styles may have surprising results. Figure 11.18 shows the markup from Figure 11.17 with some the following styles applied:

Input ▼

```
<style type="text/css" media="screen">
        fieldset {
                border: none;
                background-color: #aaa;
                width: 400px;
        }
        legend {
                text-align: right;
        }
</style>
```

Output ▶

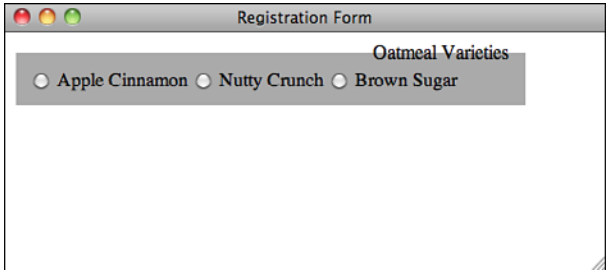


FIGURE 11.18
A fieldset and legend with styles applied.

As you can see, I've changed the background color of the field set and assigned a specific width. I've also aligned the legend to the right. Because of the default browser positioning of the legend, the background color splits the legend text. This is an example of how browsers treat legends uniquely. To set a background for the field set that includes the full legend, I'd have to wrap the field set in another block level element (like a `div`), and apply the background color to that.

Changing the Default Form Navigation

In most browsers, you can use the Tab key to step through the form fields and links on a page. When filling out long forms, it's often much easier to use the Tab key to move from one field to the next than to use the mouse to change fields. If you have a mix of form fields and links on your page, setting things up so that using Tab skips past the links and moves directly from one form field to the next can improve the usability of your applications greatly. To set the tab order for your page, use the `tabindex` attribute. You should number your form fields sequentially to set the order that the browser will use when the user tabs through them. Here's an example:

```
<p><label>Enter your <a href="/">name</a> <input type="text" name="username"
tabindex="1" /></label></p>
<p><label>Enter your <a href="/">age</a> <input type="text" name="age"
tabindex="2" /></label></p>
<p><input type="submit" tabindex="3" /></p>
```

When you tab through this page, the browser will skip past the links and move directly to the form fields.

Using Access Keys

Access keys also make your forms easier to navigate. They assign a character to an element that moves the focus to that element when the user presses a key. To add an access key to a check box, use the following code:

```
<p>What are your interests?</p>
<label>Sports <input type="checkbox" name="sports" accesskey="S" /></label>
<label>Music <input type=""checkbox" name="music" accesskey="M" /></label>
<label>Television <input type=""checkbox" name="tv" accesskey="T" /></label>
```

Most browsers require you to hold down a modifier key and the key specified using `accesskey` to select the field. On Windows, both Firefox and Internet Explorer require you to use the Alt key along with the access key to select a field. Access keys are mostly useful for forms that will be used frequently by the same users. A user who is going to use a form only once won't bother to learn the access keys, but if you've written a form for data entry, the people who use it hundreds of times a day might really appreciate the shortcuts.

Creating `disabled` and `readonly` Controls

Sometimes you might want to display a form control without enabling your visitors to use the control or enter new information. To disable a control, add the `disabled` attribute to the form control:

```
<label for="question42">What is the meaning of life?</label>
<textarea name="question42" disabled="disabled">
Enter your answer here.
</textarea>
```

When displayed in a web browser, the control will be dimmed (a light shade of gray) to indicate that it's unavailable.

To create a read-only control, use the `readonly` attribute:

Input ▼

```
<label>This month</label> <input type="text" name="month" value="September"
readonly="readonly" />
```

The read-only control is not distinguished in any way from a normal form control. However, when visitors attempt to enter new information (or, in the case of buttons or check boxes, select them), they'll find that they cannot change the value. Figure 11.19 shows both a disabled control and a read-only control. You'll generally find `disabled` to be more useful because it's less confusing to your users.

Output ▶

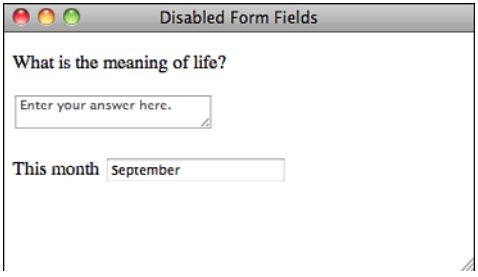


FIGURE 11.19
Disabled controls are dimmed. Read-only controls appear normally—they just can't be changed.

Form Security

It's important to remember that regardless of what you do with your form controls, what gets sent back to the server when the form is submitted is really up to your user. There's nothing to stop her from copying the source to your form, creating a similar page on her own, and submitting that to your server. If the form uses the `get` method, the user can just edit the URL when the form has been submitted.

The point here is that there is no form security. In Lesson 15, "Using JavaScript in Your Pages," you'll learn how to validate your forms with JavaScript. Even in that case, you can't guarantee that users will supply the input that you intend. What this means is that you must always validate the data entered by your users on the server before you use it.

Applying Cascading Style Sheet Properties to Form Elements

In this lesson, I've already showed you some approaches you can take to managing the layout of your forms with CSS. Now I explain how to alter the appearance of form input fields themselves using style properties. As you can see from the screenshots so far, regular form controls might not blend in too well with your pages. The default look and feel of form controls can be altered in just about any way using CSS. For example, in many browsers, by default, text input fields use Courier as their font, have white backgrounds,

and beveled borders. As you know, `border`, `font`, and `background-color` are all properties that you can modify using CSS. In fact, the following example uses all those properties:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Style Sheet Example</title>
<style type="text/css">
        input.styled
        {
          border: 2px solid #000;
          background-color: #aaa;
          font: bold 18px Verdana;
          padding: 4px;
        }
</style>
</head>
<body>
<form>
<p><label>Default</label> <input value="value" /></p>
<p><label>Styled</label> <input value="value" class="styled" /></p>
</form>
</body>
</html>
```

The page contains two text input fields: one with the default look and feel, and another that's modified using CSS. The page containing the form appears in Figure 11.20.

Output ▶

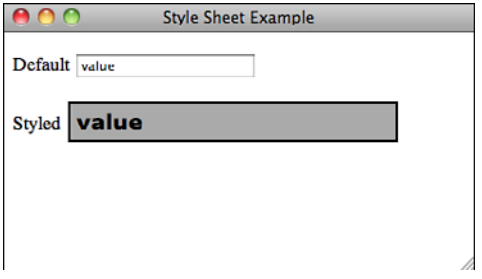


FIGURE 11.20
A regular text input field and a styled text input field.

As you can see, the field that we applied styles to is radically different from the one that uses the default decoration. You can do anything to regular form fields that you can do to other block-level elements. In fact, you can make form fields look just like the rest of your page, with no borders and the same fonts as your page text if you like. Of course,

that will make your forms extremely confusing to use, so you probably don't want to do it, but you could.

It's also fairly common to modify the buttons on your pages. Normally, Submit buttons on forms are gray with beveled edges, or they have the look and feel provided by the user's operating system. By applying styles to your buttons, you can better integrate them into your designs. This is especially useful if you need to make your buttons smaller than they are by default. I provide more examples of style usage in forms in Exercise 11.3.

Bear in mind that some browsers support CSS more fully than others. So some users won't see the styles that you've applied. The nice thing about CSS, though, is that they'll still see the form fields with the browser's default appearance.

Task: Exercise 11.3: Applying Styles to a Form

Let's take another look at the form from Exercise 11.2. The form can easily be further spruced up by tweaking its appearance using CSS. The main objectives are to make the appearance of the controls more consistent, and to make it clear to users which form fields are required and which are not. In the original version of the form, the labels for the required fields were bold. We keep with that convention here and also change the border appearance of the fields to indicate which fields are required and which aren't.

Let's look at the style sheet. This style sheet is similar to the one in Exercise 11.2, but I have made some changes. First, three styles that I copied directly from the previous exercise:

```
form div {
        margin-bottom: 1em;
}

div.submit input {
        margin-left: 165px;
}

label.field {
        display: block;
        float: left;
        margin-right: 15px;
        width: 150px;
        text-align: right;
}
```

These styles set up the basic form layout that I'm using in both exercises. Next, I tweak the appearance of my input fields:

```
input[type="text"], select, textarea {
  width: 300px;
  font: 18px Verdana;
  border: solid 2px #666;
  background-color: #ada;
}
```

The rule above applies to three different selectors: `select`, `textarea`, and `input[type="text"]`. The third selector is a bit different than the ones you've seen thus far. It is what's known as an attribute selector, and matches only `input` tags with the value of `text` for the `type` attribute. This sort of selector can be used for any attribute. In this case, I'm using it to avoid applying this rule to Submit buttons, radio buttons, and check boxes. One catch is that the attribute has to exist, so I had to add `type="text"` to my `<input>` tag. The selector won't match if you leave out the attribute and go with the default value.

Next, I add more styles that are related to the required fields. In the previous exercise, I applied the required class to the labels but I've moved it out to the `<div>`s this time around, so that I can apply it to my labels and to my form fields. The labels are still bolded, but now I use the nested rule shown next. Also note that I apply the style only to `label.required` rather than to `label`. That's so that the other labels (used for radio buttons and check boxes) aren't bolded.

```
div.required label.field {
        font-weight: bold;
}

div.required input, div.required select {
        background-color: #6a6;
        border: solid 2px #000;
        font-weight: bold;
}
```

Finally, I have made some enhancements that make it clearer which fields are required. In the original form the labels for required fields were displayed in boldface. In this example, that remains the case. However, I moved the `required` class to the enclosing div so that I can also use it in selectors that match the form fields themselves. I also styled required input fields and select fields with a dark green background color, bold type, and a different color border than optional fields have. After the style sheet is set up, all we have to do is make sure that the `class` attributes of our tags are correct. The full source code for the page, including the form updated with classes, follows:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Registration Form</title>
<style type="text/css" media="screen">
        form div {
                margin-bottom: 1em;
        }

        div.submit input {
                margin-left: 165px;
        }

        label.field {
                display: block;
                float: left;
                margin-right: 15px;
                width: 150px;
                text-align: right;
        }

        input[type="text"], select, textarea {
          width: 300px;
          font: 18px Verdana;
          border: solid 2px #666;
          background-color: #ada;
        }

        div.required label.field {
                font-weight: bold;
        }

        div.required input, div.required select {
                background-color: #6a6;
                border: solid 2px #000;
                font-weight: bold;
        }
</style>
</head>
<body>
<h1>Registration Form</h1>

<p>Please fill out the form below to register for our site. Fields
with bold labels are required.</p>

<form action="/register" method="post" enctype="multipart/form-data">

        <div class="required">
```

```
                <label class="field" for="name">Name</label>
                <input name="name" type="text" />
        </div>

        <div class="required">
                <label class="field">Gender</label>
                <label><input type="radio" name="gender" value="male" />
male</label>
                <label><input type="radio" name="gender" value="female" />
female</label>
        </div>

        <div class="required">
                <label class="field">Operating System</label>

                <select name="os">
                        <option value="windows">Windows</option>
                        <option value="macos">Mac OS</option>
                        <option value="linux">Linux</option>
                        <option value="other">Other ...</option>
                </select>
        </div>

        <div>
                <label class="field">Toys</label>
                <label><input type="checkbox" name="toy" value="digicam" />
Digital Camera</label>
                <label><input type="checkbox" name="toy" value="mp3" /> MP3
Player</label>
                <label><input type="checkbox" name="toy" value="wlan" />
Wireless LAN</label>
        </div>

        <div>
                <label class="field">Portrait</label>
                <input type="file" name="portrait" />
        </div>

        <div>
                <label class="field">Mini Biography</label>
                <textarea name="bio" rows="6" cols="40"></textarea>
        </div>

        <div class="submit">
                <input type="submit" value="register" />
        </div>
</form>
</body>
</html>
```

Figure 11.21 shows the page containing this form.

Output ▶

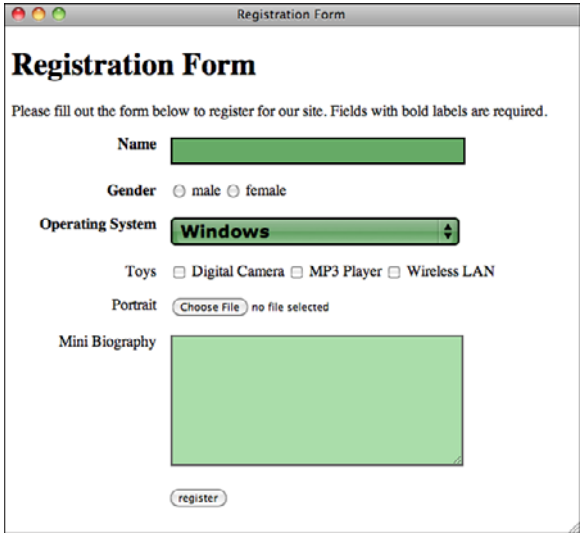


FIGURE 11.21
A form with styled input fields.

Planning Your Forms

Before you start creating complex forms for your web pages, you should do some planning that will save you time and trouble in the long run.

First, decide what information you need to collect. That might sound obvious, but you need to think about this before you start worrying about the mechanics of creating the form.

Next, review this information and match each item with a type of form control. Ask yourself which type of control is most suited to the type of questions you're asking. If you need a yes or no answer, radio buttons or check boxes work great, but the `textarea` element is overkill. Try to make life easier for the users by making the type of control fit the question. This way, analyzing the information using a script, if necessary, will be much easier.

You also need to coordinate with the person writing the CGI script to match variables in the script with the names you're assigning to each control. There isn't much point in naming every control before collaborating with the script author—after all, you'll need all the names to match. You also can create lookup tables that contain expansive descriptions and allowable values of each form control.

Finally, you might want to consider validating form input through scripting. Using JavaScript, you can embed small programs in your web pages. One common use for JavaScript is writing programs that verify a user's input is correct before she submits a form. I discuss JavaScript in more detail in Lesson 14.

Summary

As you can see, the wonderful world of forms is full of different types of form controls for your visitors. This truly is a way to make your web pages interactive.

Be cautious, however. Web surfers who are constantly bombarded with forms are likely to get tired of all that typing and move on to another site. You need to give them a reason for playing!

Table 11.1 summarizes the HTML tags used today. Remember these points and you can't go wrong:

- Use the `form` element to create your forms.
- Always assign an `action` to a form.
- Create form controls with the `input` element or the other form control elements.
- Test your forms extensively.

TABLE 11.1 HTML Tags Used in this Lesson

Tag	Use
`<form>`	Creates an HTML form. You can have multiple forms within a document, but you cannot nest the forms.
`action`	An attribute of `<form>` that indicates the server-side script (with a URL path) that processes the form data.
`enctype`	An attribute of the `<form>` tag that specifies how form data is encoded before being sent to the server.
`method`	An attribute of `<form>` that defines how the form data is sent to the server. Possible values are `get` and `post`.
`<input>`	A `<form>` element that creates controls for user input.
`type`	An attribute of `<input>` that indicates the type of form control. Possible values are shown in the following list:
`text`	Creates a single-line text entry field.
`password`	Creates a single-line text entry field that masks user input.
`submit`	Creates a Submit button that sends the form data to a server-side script.

TABLE 11.1 Continued

Tag	Use
`reset`	Creates a Reset button that resets all form controls to their initial values.
`checkbox`	Creates a check box.
`radio`	Creates a radio button.
`image`	Creates a button from an image.
`button`	Creates a pushbutton. The three types are Submit, Reset, and Push, with no default.
`hidden`	Creates a hidden form control that cannot be seen by the user.
`file`	Creates a file upload control that enables users to select a file with the form data to upload to the server.
`<button>`	Creates a button that can have HTML content.
`<textarea>`	A text-entry field with multiple lines.
`<select>`	A menu or scrolling list of items. Individual items are indicated by the `<option>` tag.
`<option>`	Individual items within a `<select>` element.
`<label>`	Creates a label associated with a form control.
`<fieldset>`	Organizes form controls into groups.
`<legend>`	Displays a caption for a `<fieldset>` element.

Workshop

If you've made it this far, I'm sure that you still have a few questions. I've included a few that I think are interesting. Afterward, test your retention by taking the quiz, and then expand your knowledge by tackling the exercises.

Q&A

Q **Are there security issues associated with including forms on my website?**

A Yes and no. The forms themselves are not a security risk, but the scripts that process the form input can expose your site to security problems. Using scripts that you can download and use on your own site can be particularly risky, because malicious people will already know how to exploit any of their bugs. If you are going to use publicly available scripts, make sure they are approved by your hosting provider and that you are using the latest release.

Q I want to create a form and test it, but I don't have the script ready. Is there any way I can make sure that the form is sending the right information with a working script?

A I run into this situation all the time! Fortunately, getting around it is very easy.

Within the opening `<form>` tag, modify the `action` attribute and make it a `mailto` link to your email address, as in the following:

```
<form action="mailto:youremailaddress@isp.com" method="post">
```

Now you can complete your test form and submit it without having a script ready. When you submit your form, it will be emailed to you as an attachment. Just open the attachment in a text editor, and presto! Your form data is present.

Quiz

1. How many forms can you have on a web page?
2. How do you create form controls such as radio buttons and check boxes?
3. Are passwords sent using a `password` control secure?
4. Explain the benefit of using hidden form controls.
5. What other technology do forms rely on?

Quiz Answers

1. You can have any number of forms on a web page.
2. These form controls are created with the `input` element. Radio buttons have the `type` attribute set to `radio`, and check boxes are created using the type `checkbox`.
3. No! Passwords sent using a `password` control are not secure.
4. Hidden form controls are intended more for you than for the person filling out the form. By using unique `value` attributes, you can distinguish between different forms that may be sent to the same script or sent at different times.
5. For you to process the data submitted via forms, they must be paired with a server-side script through the `action` attribute.

Exercises

1. Ask your hosting provider for scripts that you can use to process your forms. If you can use them, ask how the data is processed and which names you should use in your form controls. If you need to use forms and your hosting provider won't allow you to use its scripts, start looking elsewhere for a place to host your website.
2. Visit some sites that might use forms, such as http://www.fedex.com. Look at which form controls they use and how they arrange them, and peek at the source to see the HTML code.

This page intentionally left blank

LESSON 12

Integrating Multimedia: Sound, Video, and More

Video and sound are a core part of the modern Web. You can watch television online at sites like Hulu, watch movies on demand through Netflix and Amazon.com, and watch videos uploaded by anyone at sites like Vimeo and YouTube. Sites that sell downloadable music provide audio previews of the music they sell. Pandora and Last.fm enable their users to create their own radio stations starting with the name of a single song.

Understanding how to incorporate video, audio, and Flash into your own Web pages is an important part of building modern websites. In this lesson you'll learn how to:

- Embed videos hosted at Vimeo and YouTube in your pages
- Convert video files to common Web formats
- Use the HTML5 `<video>` and `<audio>` tags
- Embed Flash movies in Web pages using the `<object>` tag
- Use Flash audio and video players

Embedding Video the Simple Way

There's a lot to be learned about embedding video: tags for embedding video in web pages, new audio and video elements in HTML5, and browser incompatibilities. First, however, it's worth discussing the way the vast majority of videos are embedded in web pages these days. People upload their video to a website that specializes in video hosting or use videos that other people have uploaded and then copy and paste the code those sites provide for embedding the videos in their own sites.

The two most popular video hosting sites, YouTube and Vimeo, provide the code to embed the videos they host on the web pages for each video. You can see the form that allows you to generate the embed code for YouTube in Figure 12.1.

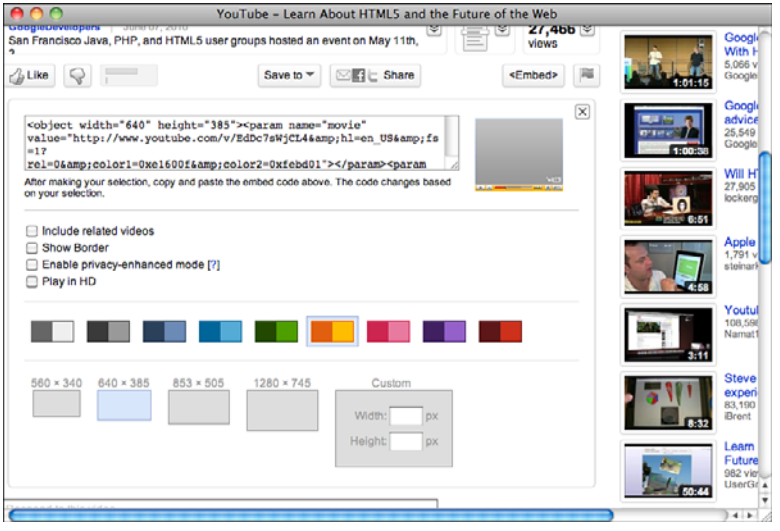


FIGURE 12.1
The embed form on YouTube.

If you want to add a video hosted on YouTube, you just click the Embed button and the form shown in the figure displays. Choose your customization options, and then copy the HTML code from the box into your own page. The result is shown in Figure 12.2.

YouTube (and other sites) automatically generate the markup that you can paste into your own web pages to embed their videos. There's nothing magic about it; I'll describe the tags they use when I discuss embedding video later.

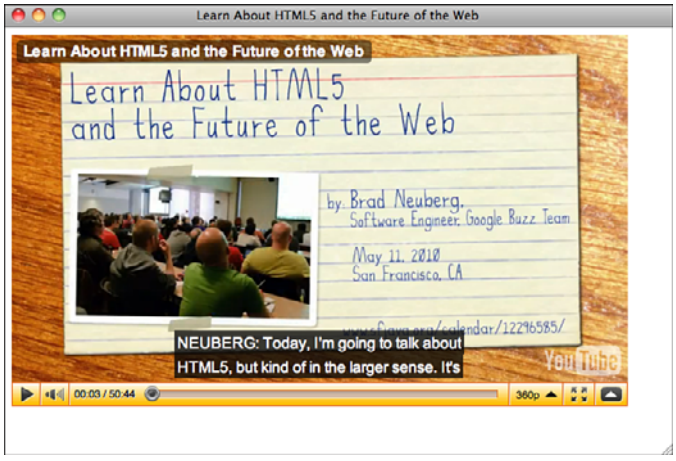


FIGURE 12.2
A YouTube video embedded in a web page.

Advantages and Disadvantages of Hosting Videos on YouTube

You can upload your own videos to YouTube and embed them in your pages, too. Other sites, like Vimeo (http://vimeo.com/) also offer free hosting for video. There are a number of advantages to hosting your videos on YouTube. For one thing, video files tend to be rather large, and hosting them on YouTube means that you don't have to figure out a place to put them. You also get to take advantage of YouTube's video player, which supports multiple quality levels and full-screen playback. It's used by millions of people and is widely tested. There's are also YouTube applications for mobile platforms like Apple iOS and Google Android, so YouTube videos can be viewed on them, whereas they cannot be with other Flash players. As you'll see, another advantage is that it's easy to start with YouTube. You just upload your video file, go to the new page for the video, and then copy the embed code and paste it on your own site to get things working.

Another advantage of hosting your video on YouTube is that you can take advantage of YouTube's audience in addition to the audience at your own website. When you upload a video to YouTube and make it public, it shows up in search results and on the lists of related videos when people watch other videos on the site. So in the end, using YouTube for video hosting can lead more people to your website than hosting videos on your own.

The disadvantages of using YouTube are that you cede some control over your video and how it is presented. The YouTube player works well, but it's obvious to your users that it's the YouTube player. You may want to host the video yourself if you don't want them to be distracted by YouTube, or if you want to use your own player.

Uploading Videos to YouTube

YouTube provides a number of ways to upload video to the site. You can take video with your webcam and upload it directly, or even send video taken on a smart phone to YouTube. In this case, I'm going to upload an existing video file using the web interface. To start the process, go to http://upload.youtube.com/my_videos_uploada and click the Upload Video button. When you do so, a file selection box will appear that you can use to locate and select the video file you want to upload.

NOTE

YouTube supports a wide variety of video formats, including those used by most camcorders. Supported formats include MP4, MOV, AVI, MWV, and FLV.

After you've selected a file, you'll immediately be taken to the file upload page, shown in Figure 12.3. The page shows a progress indicator that lets you know how long your video is going to take to upload and enables you to enter information about the video you've uploaded. Using the form you can enter a title, description, category, and tags for your video, all of which are important if you want YouTube users to find your video.

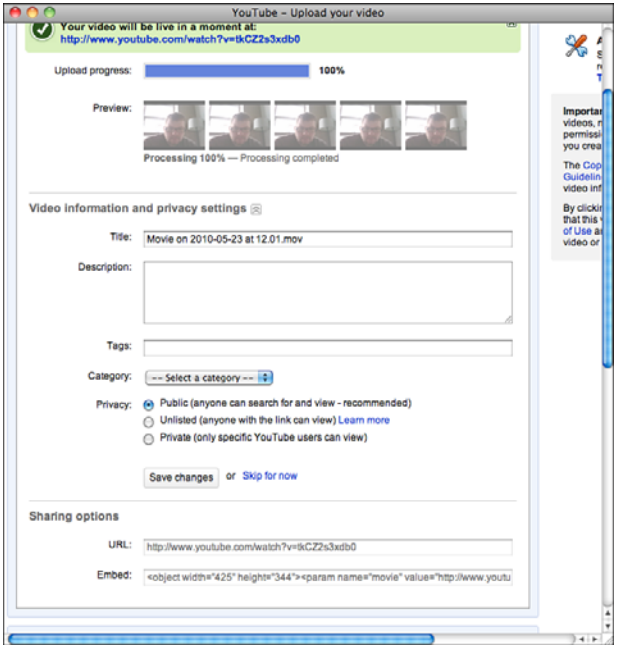


FIGURE 12.3
The YouTube file upload page.

As you can see from the screenshot, you can also choose a privacy setting for your video. You can choose public, which allows people to find your video through YouTube, or private, meaning that you can specify exactly who's allowed to see it. The third option is unlisted. This option makes the video publicly available, but only to people who know the URL. It's useful if you want to embed the video on your own website, but you don't want people to find it by browsing YouTube.

YouTube provides the URL and embed code for your video before it's even finished uploading, so you can link to it immediately.

Customizing the Video Player

After you've uploaded your video, you can embed it in your own web pages. Embedding videos of your own is just like embedding other videos found on YouTube, you can just click the Embed button and copy the code for your own page. However, you can do some things to customize the embedded player. You can see all the embedding options in Figure 12.4.

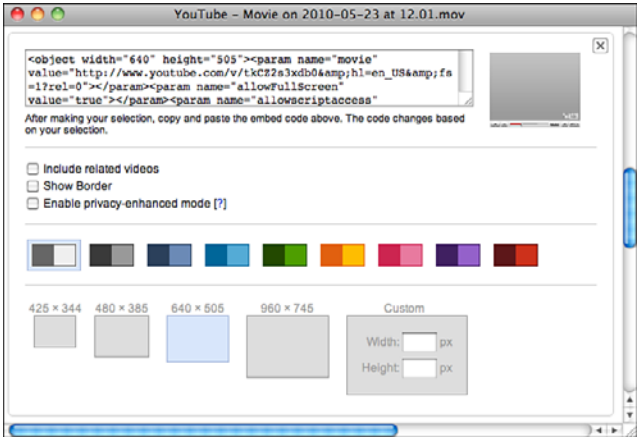


FIGURE 12.4
Customization options for embedded YouTube videos.

As you tweak the embed settings, the page automatically updates the embed code with your new settings. There are three check boxes you can select. The first allows you to disable the list of related videos that YouTube normally displays when a video finishes playing. You may want to disable these if you want your visitors to stick around on your site after watching your video instead of wandering off to look at other videos on YouTube. Enabling Show Border adds a border to the YouTube player (and adjusts the height and width to accommodate the border without shrinking the video). Privacy-enhanced mode prevents YouTube from storing identifying information about the user if they don't click the player.

The remaining options enable you to alter the appearance of the video player. You can adjust the color scheme and the height and width of the video from this page. After you've chosen all of your customization options, you can copy the embed code and use it in your page. Later in this lesson, you'll learn about what these customization options change in the underlying markup.

Other Services

YouTube is the most popular video hosting service, but there are many others, too. Vimeo (http://vimeo.com/) is a popular video hosting service that's a lot like YouTube. YouTube offers unlimited uploads but limits the length of video uploads to 10 minutes. Vimeo offers a professional (paid) account that enables subscribers to upload videos of any length. Many people also think that Vimeo's video player, shown in Figure 12.5, is more attractive than YouTube's because the controls disappear entirely when you're watching the video.

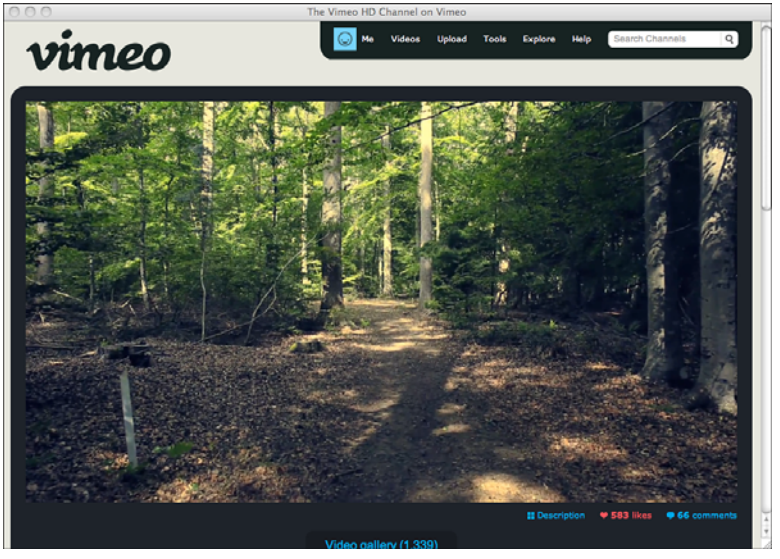


FIGURE 12.5
Vimeo's video player.

The process of uploading video files to Vimeo is nearly identical to the process for YouTube. You just choose your file and information like the name and description. Both sites will convert video from nearly any common format to the format used by their player.

Here's a list of some other popular video hosting services:

- **blip.tv**—http://blip.tv/
- **Dailymotion**—http://www.dailymotion.com/
- **Flickr**—http://flickr.com/
- **SmugMug**—http://www.smugmug.com/
- **Viddler**—http://www.viddler.com/
- **VideoPress**—http://videopress.com/

Which video hosting site you choose is a matter of taste and ease of use. Each site has its own video player and its own community, and you should choose whichever suits you best. Be sure to check out the restrictions on video length and video resolution when choosing. For example, the maximum length of videos on Flickr is 90 seconds, and only users with Pro accounts are allowed to view them in high definition (HD). There's also no rule that says that you can't upload your videos to more than one site. You may want to upload your videos to Vimeo for the purpose of embedding them on your own site and upload them to YouTube to make them available to YouTube's audience.

Hosting Your Own Video

For any number of reasons, you might want to host video yourself instead of relying on a third-party service such as Vimeo or YouTube to host it for you. For one thing, you can use your own player rather than using the one they provide. You also may not want to include branding or advertising from a third party on your own site, and you might not want to distract your users with a link to YouTube. As is the case with most third-party services on the Web, hosting your own video gives you more control over the end result but requires more work and expertise on your part.

At one time, a wide variety of methods were used to embed video in web pages, each with its own browser plug-in and file format. These days, just two common methods are in use. The first is to use a Flash movie to play back the video, and the second is to use the HTML5 `<video>` tag to play the video using the browser itself. I'll explain how to use both approaches, and how to combine the two to support as many browsers and platforms as possible.

Before diving into the tags used to publish video on the Web, it's important to first explain how to create video files that can be played in a browser. Understanding how to create video files for the Web is the first step in getting video from your camcorder or mobile phone onto web pages.

Video and Container Formats

Before discussing how to embed video within a web page, it's important to discuss video formats. All video files are compressed using what's known as a codec, short for coder/decoder. After a video has been encoded, it must be saved within a container file, and just as there are a number of codecs, there are a number of container file formats, too. To play a video, an application must understand how to deal with its container file and decode whatever codec was used to compress the video. For example, H.264 is one of the most popular video codecs and is supported by a number of container formats, including FLV (Flash Video) and MP4.

It's not uncommon to run into situations where a video player can open the container file used to package the video but does not support the codec used to encode the video. Likewise, if a video player doesn't recognize the container file used to package the video, it won't play it back, regardless of the codec used. Whereas many, many video codes and container formats exist, only a few are relevant in terms of video on the Web. The extension for a video file indicates its container format, not the codec of the video in it. For example, the extension for Apple's QuickTime container format is `.mov`, regardless of which codec is used to encode the video.

H.264 is a commercial format that is supported natively by Microsoft Internet Explorer 9, Apple Safari, and Google Chrome. It's also supported by Flash. The problem with H.264 is that it is patented, and there are license fees associated with the patents. Companies that implement the codec must pay for a license as must companies that use the codec to deliver H.264 video to users. Mozilla has decided not to support H.264 in Firefox because of the patent licenses required. H.264 is the most popular format for delivering video content over the Web by far. It's also used for satellite and cable television and to encode the video on Blu-ray discs.

Most commonly, H.264 video is associated with MP4 (`.mp4`) containers or occasionally Flash Video (`.flv`) containers. MP4 files are supported by the Flash player and by all the browsers that support H.264 video, making it the most widely supported container for distributing video on the Web.

Theora is an open, freely licensed video codec released by the Xiph.org Foundation. Mozilla Firefox and Google Chrome offer Theora support, but Apple and Microsoft have no plans to support it. It's usually associated with the Ogg container format, and the files are usually referred to as Ogg Theora files. Ogg files that contain video usually have the extension `.ogv`. There's also an associated audio codec, Vorbis. Ogg Theora files have the extension `.oga`.

In 2010, Google released a new container format called WebM. WebM files use the VP8 codec for video and the Vorbis codec for audio. VP8 was originally created by a company called On2, which was acquired by Google, who then released the codec to the public without any licensing requirements. WebM is supported by Google Chrome and will also be supported by Mozilla Firefox, Microsoft Internet Explorer, and Adobe Flash.

Currently, if you want to encode your video only once, you can use H.264/MP4 and play it natively in browsers that support it using the `<video>` tag. Other browsers can play the same video file using a Flash video player. If you want to avoid using a Flash-based player, the safest bet is to encode your video using Ogg Theora, as well, so that it will play in Firefox.

Converting Video to H.264

One of the nicest features of video hosting services is that they free you from worrying about codecs and container formats, because they do the conversion for you. It's up to you to create a video file with the desired resolution, but they take it from there. If you're hosting video yourself, you'll need to convert your video to MP4 and perhaps Ogg Theora, too.

A number of tools are available for dealing with video, but when it comes to converting video from other formats to H.264, there's only one you need to worry about: Handbrake. Handbrake is a free, open source application that enables you to convert video stored in pretty much any format to H.264. There are versions for Windows, OS X, and Linux that all work basically the same. You can download the Handbrake at http://handbrake.fr/.

If you just want to convert your video to H.264, you can open it in HandBrake and click Start. However, you'll probably want to tweak some of the settings to optimize your video for use on the Web. Check out the interface for HandBrake in Figure 12.6. I'll walk you through the options you'll want to set to optimize your video for the Web.

First, choose a filename for your video using the Destination field. You'll also want to stick with the default output format, MP4. The four tabs at the bottom enable you to optimize the video output for your purpose. First, though, check the Web Optimized button for your video. It enables your video to start playing immediately as it's being downloaded and makes it easier for players to skip around in the video. The only cost is slightly longer encoding time.

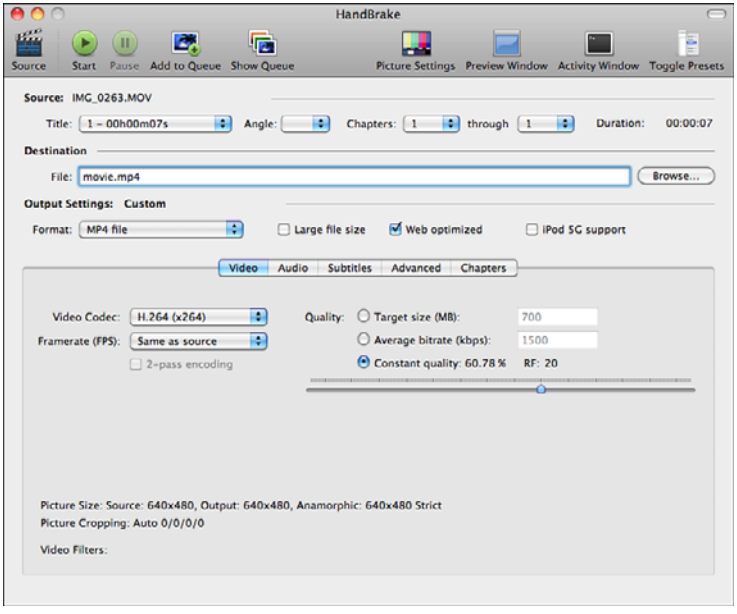


FIGURE 12.6
The interface for HandBrake.

Under the video options, the default codec is H.264. Keep that. Under Framerate, the default is to stick with the framerate in the video you're converting, but you can choose another option. The higher the framerate, the larger the resulting file. If you change the framerate, you can enable two-pass encoding, which causes encoding to take longer (by adding the additional pass), but results in higher-quality video for a given file size.

Finally, you'll tweak the Quality settings. Video encoding is all about trade-offs. The higher the picture quality, the larger the resulting file. Larger files take up more space on the server and take longer to download. On the other hand, they look better. There are three variables you can change that affect the overall size of the file—the height and width of the video (a 320x240 video will be much smaller than a 640x480 video), the framerate, and the quality. If your video will be played in a small box embedded on a web page, you can afford to lower these settings to create smaller videos. If your video will be played on a 42-inch television, you'll probably want to raise the quality settings.

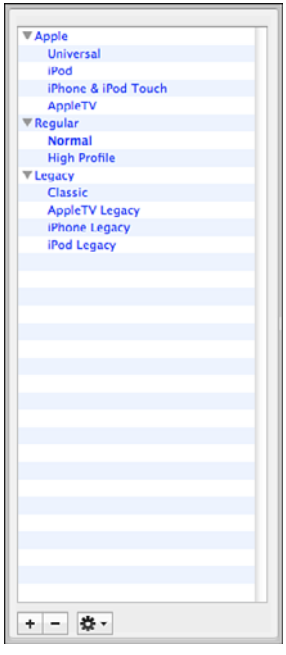
There are three ways to specify the quality for your video, and understanding them requires that you know about bit rate. The bit rate of a video is the amount of data used by one second of video. The bigger the number, the more space the video will use. The default method of specifying quality is "constant quality." What this means is that the entire video will be compressed by the same factor. H.264 is like the JPEG image format, in that some data is lost when the video is compressed. The Constant Quality setting

applies the same compression factor to the whole video. When you set a video to Constant Quality, the video uses whatever bit rate is required to provide that quality, so the bit rate will vary throughout depending on how well the video can be compressed at the specified quality level.

The Average Bit Rate option varies the quality of the video to satisfy the bit rate that you specify. Both it and the target file size option are more predictable—the video will be the size you expect when you get to the end.

Instead of manipulating the settings on your own, you can use one of the presets that HandBrake provides. To view the presets, click the Toggle Presets button on the upper right. The list of settings in Figure 12.7 will appear. (I've fully expanded the list.) These presets are already optimized for certain uses. The one that works best for web video is the iPhone & iPod Touch preset. The simplest approach, if you're starting out, is to select it and then click the Web Optimized check box.

FIGURE 12.7
The list of HandBrake presets.



You'll also want to click the Picture Settings button in the toolbar to specify the height and width of your video. 320x240 is a pretty standard size for smaller videos. 640x480 is also a common setting. For HD video, set the size to at least 1280x720. From Picture Settings, you can also crop your video or adjust the filter settings. The HandBrake documentation has more on the filter settings.

After you've specified the settings, just click the Start button to encode your video as H.264. When the encoding is complete, preview the video, preferably in the player you'll be using on the Web, to make sure that the quality is sufficient. If it's not, encode the video again using different settings. Likewise, if the video file is larger than you'd like, you may want to encode the video again with the compression turned up. Afterward, watch the video and make sure that it still looks okay.

Converting Video to Ogg Theora

To convert video for use with Firefox and Chrome, you'll need to convert it to Ogg Theora. A number of tools can be used to do so. One popular, free option is Firefogg, which can be found at http://firefogg.org/. Firefogg is an add-on for the Firefox web browser. When it's installed, you can go back to the Firefogg website to convert video from other formats to Ogg Theora.

VLC, a video player that supports a wide variety of formats, can also be used to convert video to Ogg Theora. It's available for Windows, OS X, and Linux. You can download it at http://www.videolan.org/vlc/. To use it to convert video from one format to another, use the Open Capture Device option in the File menu, select the file you want to convert, and then use the Transcoding Options to choose the Theo video format and Vorb audio format.

Embedding Video Using `<video>`

The methods used to embed video in web pages have changed a great deal over the years. In the early days of the Web, to present video, the best approach was just to link to video files so that users could download them and play them in an application other than their browser. When browsers added support for plug-ins through the `<embed>` tag, it became possible to embed videos directly within web pages. The catch was that to play the video, the user was required to have the proper plug-in installed.

The tag used to embed plug-ins in pages changed from `<embed>` to `<object>`, but the approach was the same. Plug-ins made embedding videos possible, but they didn't make it easy, because of the wide variety of video formats and plug-ins available. Publishing video files that worked for all, or even most, users was still a problem.

In 2002, Adobe added support for video to Flash. Because nearly everyone had Flash installed, embedding videos in Flash movies became the easiest path to embedding video in web pages. Later, it became possible to point a generic Flash video player at a properly encoded movie and play it. This is how sites like YouTube work. As you'll see later in this lesson, there are some Flash video players that you can use to play videos that you

host, too. Currently, Flash remains the most popular approach for presenting video on the web, but with HTML5, browsers are beginning to add native support for video playback through the <video> tag.

The current generation of mobile devices that are capable of video playback (like the iPhone and phones based on Google's Android operating system) support the HTML5 <video> tag and in most cases do not support Flash. So, the best approach for providing video to the widest number of users is to use both the <video> tag and a Flash player. After introducing the <video> tag, I'll explain how to use it with a Flash movie in such a way that users only see one video player—the appropriate one for their environment.

The <video> Tag

The <video> tag is new in HTML5. It is used to embed a video within a web page, and to use the browser's native video playback capabilities to do it, as opposed to Flash or some other plug-in. Here's a simple version of the <video> tag:

```
<video src="myvideo.mp4">
```

If the browser is capable of playing the video at the URL specified in the src attribute, it will do so. Or it would, if there were some way of telling the browser to play the video. In this case, the video will have no controls and won't start playing automatically. To take care of that, I need to use some of the attributes of the <video> tag, which are listed in Table 12.1.

TABLE 12.1 <video> Attributes

Attribute	Description
src	The URL for the video to be played.
height	The height of the element.
width	The width of the element.
controls	Boolean attribute that indicates that the browser should supply its own controls for the video. The default is to leave out the controls.
autoplay	Boolean attribute that indicates that the video should play immediately when the page loads.
loop	Boolean attribute. If present, the video will loop when it reaches the end, replaying until the user manually stops the video from playing.
preload	Boolean attribute. If present, the browser will begin downloading the video as soon as the page loads to get it ready to play. Ignored if autoplay is present.

Because the video above doesn't have the `controls` or `autoplay` attributes, there's no way to get it to play. Figure 12.8 shows the video, embedded using the following tag, with `controls` included:

```
<video src="myvideo.mp4" controls>
```

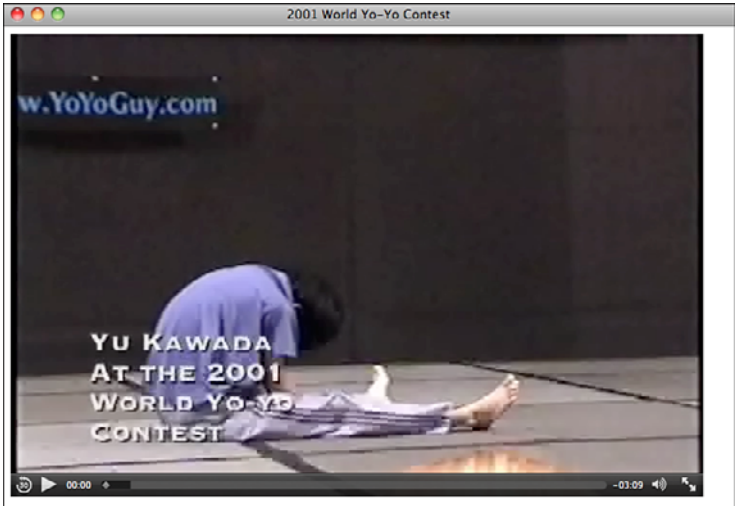


FIGURE 12.8
A video embedded using the `<video>` tag, with `controls`.

CAUTION

When embedding a video, make sure that you give users some way to control the video playback. Be conservative with `autoplay` and `loop`, too. Many users don't want a video to start playing immediately when they get to a page. If you include the `loop` attribute and you don't include any controls, the user will have to leave the page to stop the video.

By default, the `<video>` element will be the same size as the video in the video file. You can change the size of the element on the page using the `height` and `width` attributes; however, the browser will preserve the aspect ratio of the video file and leave blank space where needed. For example, my movie was created using a 3:2 aspect ratio. If I create a `<video>` element with a 9:5 aspect ratio, the movie will appear centered within the element, as shown in Figure 12.9:

```
<video style="background: black" src="http://www.yo-yo.org/mp4/yu.mp4" controls
width="675" height="375">
```

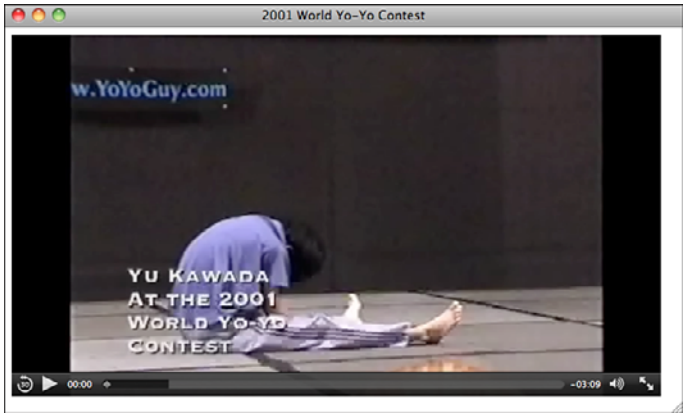


FIGURE 12.9
A <video> tag with a different aspect ratio than the embedded video file.

I set the background color of the <video> element to black to make it clear where the browser puts the extra space when the movie's aspect ratio does not match the aspect ratio of the element.

Finally, if you're fairly certain that most people who come to your page will want to view the video, but you want to let them play the video themselves, you may want to include the preload attribute, which tells the browser to go ahead and download the video when the page loads but to wait for the user to play the video. Usually this means users will not have to wait as long to see the video after they try to play it, but the disadvantage is that you'll use bandwidth sending the video to everyone, whether they actually play the movie.

Using the <source> Element

There are two major drawbacks to using the <video> tag. The first is that all browser support is limited. Versions 8 and earlier of Internet Explorer do not offer support for <video>, nor does Firefox 3.0. The second is that even among browsers that support the <video> tag, not all of them support the same video containers or codecs. As you've seen, this problem requires you to encode your videos in multiple formats if you want to reach most browsers, but the good news is that the <video> element provides a means of dealing with this issue so that your users won't notice.

To embed a single video file, you can use the src attribute of the video tag. To provide videos in multiple formats, use the <source> element nested within your <video> tag. Here's an example, the results of which are shown in Figure 12.10:

```
<video width="320" height="240" preload controls>
       <source src="movie.mp4"
                type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

```
        <source src="movie.ogv"
                type='video/ogg; codecs="theora, vorbis"'>
</video>
```

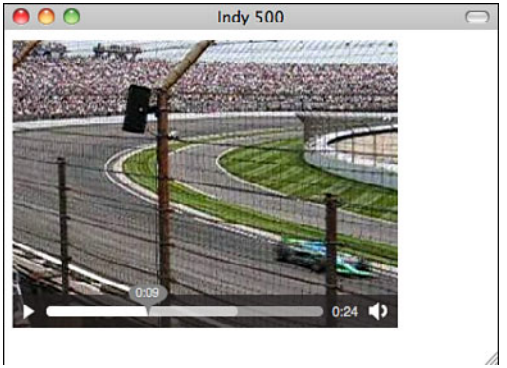


FIGURE 12.10
A video embedded using the `<video>` tag with `<source>` tags with controls.

As you can see, in this case I've left the `src` attribute out of my `<video>` tag. Instead, I've nested two `<source>` elements within the tag instead. The `src` attribute of `<source>` contains the URL a video file, and the `type` attribute provides information to the browser about the format of that file. The browser examines the types of each of the movie files and chooses one that is compatible.

The syntax of the `type` attribute can be a little bit confusing because of the punctuation. Here's the value:

```
video/ogg; codecs="theora, vorbis"
```

The first part is the MIME type of the video container. The second part lists the codes that were used to encode the audio and video portions of the file. So in this case, the container type is `video/ogg`, the video codec is `theora`, and the audio codec is `vorbis`. If the browser supports both the file type and the codecs, it will use that video file. The values for the `type` attribute are as follows:

- **MP4/H.264**—`video/mp4; codecs="avc1.42E01E, mp4a.40.2"`
- **Ogg Theora**—`theora, vorbis`
- **WebM**—`vp8, vorbis`

Embedding Flash Using the `<object>` Tag

The `<object>` tag is used to embed media of all kinds in web pages. Although it is most often used to embed Flash movies, it can also be used for audio files, video files, images, and other media types that require special players. Unlike all the other HTML tags you've learned about so far, the `<object>` tag works differently from browser to browser.

The problem is that browsers use different methods to determine which plug-in should be used to display the media linked to through the <object> tag.

First, the version of the <object> tag that works with Internet Explorer:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780"
  height="420">
  <param name="movie" value="movie.swf" />
</object>
```

The height and width attributes are necessary to define how much space the <object> will take up. The classid attribute identifies the ActiveX control that will be used to display the content in the browser. That long, random-looking collection of letters and numbers is the address of the ActiveX control in the Windows Registry. Your best bet is to find an example of Flash embedding and copy it from there.

TIP

When you're specifying the height and width for a Flash movie, be sure to take the size of the player into account, too. Some players include a border, and nearly all of them provide controls for the video playback. You need to account for these parts of the window to make sure your video is shown at the resolution you anticipated.

The <param> element is used with <object> to provide additional information about the content being embedded to the plug-in referenced by the <object> tag. The <param> element has two attributes, name and value. This <param> element provides the Flash player with the URL of the movie to be played.

The preceding markup will work in Internet Explorer, embedding the Flash movie named movie.swf. Here's the markup for the <object> tag for other browsers:

```
<object type="application/x-shockwave-flash" data="myContent.swf" width="780"
height="420">
</object>
```

For non-Internet Explorer browsers, you specify the plug-in to use with the type attribute and the URL of the movie to play with the data attribute. As you can see, the height and width attributes are included here, too. The type attribute is used to provide an Internet media type (or content type). The browser knows which content types map to which plug-ins, so it can figure out whether you have the proper plug-in installed. If you do, it can load it and render the content at the URL specified by the data attribute. In the sidebar, I explain exactly what Internet media types are.

Internet Content Types

Internet media types, also referred to as content types or MIME types, are used to describe the format of a file or resource. They're a more robust version of a file extension. For example, a PNG image usually has the extension `.png`. The MIME type for PNG files is `image/png`. Microsoft Word documents have the extension `.doc` (or more recently, `.docx`) and the MIME type `application/msword`. These types were originally used to identify the types of email attachments—MIME is short for Multipurpose Internet Mail Extensions—but these days, they're used in other cases where information about file types needs to be exchanged.

In the case of the `<object>` tag, you specify an Internet media type so that the browser can determine the best way to render the content referenced by that tag. The Internet media type for Flash is `application/x-shockwave-flash`, if that type is specified, the browser knows to use the Flash plug-in.

There's one other important use for these types when it comes to video and sound files. When a web server sends a resource to the Web, it includes a `content` type. The browser looks at the `content` type to determine what to do with the resource. For example, if the content type is `text/html`, it treats it as a web page.

When a web server sends files to users, it figures out the Internet media type using the file extension. So if a user requests `index.html`, the web server knows that an extension of `.html` indicates that the files should have the content type `text/html`. Later in this lesson, I discuss how to make sure that your web server sends the right content types for video and audio files that you use.

With most tags, you could just combine all the attributes and wind up with an `<object>` that works with all the popular browsers. With `<object>`, it doesn't work that way. However, there's a way around this problem. Here's a version that will work:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780"
height="420">
  <param name="movie" value="movie.swf" />
  <object type="application/x-shockwave-flash" data="myContent.swf" width="780"
height="420">
  </object>
</object>
```

In this example, one of the `<object>` tags is nested inside the other. This works because browsers ignore tags they don't understand, so browsers that aren't Internet Explorer ignore the outer `<object>` tag. Internet Explorer ignores tags nested inside an `<object>` tag, except for the `<param>` tag, so it ignores the inner `<object>`. That's the simplest approach to using the `<object>` tag in a way that works with all browsers.

A number of other attributes are supported by the <object> tag, too (see Table 12.2).

TABLE 12.2 <object> Attributes

Attribute	Description
align	Aligns the element to the left, right, top, or bottom. This attribute is superseded by the Cascading Style Sheets (CSS) align property and is removed from HTML5.
archive	A list of URLs for resources that will be loaded by the browser. Removed from HTML5 because it was rarely used.
border	The size of the border in pixels. Removed from HTML5 because it is superseded by the CSS border property.
classid	Specifies the class ID value of the plug-in to be used. This is a reference to the plug-in's address in the Windows Registry. Removed from HTML5.
codebase	The URL where the plug-in can be downloaded if the user does not have it installed. Only works with Internet Explorer. Removed from HTML5.
codetype	The MIME type of the code (plug-in) referenced by the codebase attribute. This isn't the type of the media, but rather of the plug-in used to play the media. Removed from HTML5 and rarely used.
data	The URL for the data that will be presented in the <object> element. Flash uses a <param> to specify this instead.
declare	Indicates that the object is only a declaration and should not be created or displayed. Rarely used and removed from HTML5.
form	New attribute in HTML5 that enables the element to be associated with a specific form.
height	The height of the element.
hspace	The horizontal padding around the object. Superseded by the padding CSS property and removed from HTML5.
name	A unique name for the element.
standby	Contains text to be displayed while the object is loading. Removed from HTML5, in favor of using CSS and JavaScript to display a progress message.
type	The MIME type of the content to be displayed in the object.
usemap	Specifies the URL of a client-side image map to be applied to the object.
vspace	The vertical padding for the object. Superseded by the padding CSS property and removed from HTML5.
width	The width of the element.

In HTML5, you may find yourself using the <video> tag rather than the <object> tag for video files, but the <object> tag will still be used for other Flash movies, and for other multimedia content, such as Microsoft Silverlight.

Alternative Content for the <object> Tag

What happens when a user hasn't installed the plug-in that the <object> tag requires? The browser will either display an error message or just display nothing at all. However, you can provide alternative content that will be displayed if the browser cannot find the correct plug-in. All you have to do is include the alternative content inside the <object> tag. If the <object> tag works, it will be ignored. If it doesn't, it will be displayed. Here are the nested <object> tags with some alternative content included. You can see alternative content displayed in a browser that does not have Flash installed in Figure 12.11. Here's the code:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780"
height="420">
  <param name="movie" value="movie.swf" />
  <object type="application/x-shockwave-flash" data="myContent.swf" width="780"
height="420">
    <p>You need the Flash player to view this page.
    <a href="http://get.adobe.com/flashplayer/>Get Flash.</a></p>
  </object>
</object>
```

FIGURE 12.11
Alternative content displayed in a browser that doesn't support Flash.

TIP

It's often a good idea to make your alternative content the same size as the <object> tag to preserve the layout of your page. You can style your alternative content with CSS or use an image of the same size as a placeholder for the <object>.

The <embed> Tag

The <embed> element has been added to HTML5, mainly as a recognition that it has been in wide use since Netscape created it when it added plug-in support to their browser. Browsers continue to support it, mainly because many pages out there still use it. The default YouTube embed code includes the <embed> tag.

First, let's look at the required attributes of the <embed> element:

```
<embed src="a01607av.avi" height="120" width="160"
  type="application/x-shockwave-flash" />
```

The src attribute contains the location of the multimedia file you want to embed in the web page. The type attribute contains the content type. (It's the same as the type attribute of the <object> tag.) The height and width attributes specify the dimensions of the embedded file in pixels.

Table 12.3 summarizes the <embed> attributes supported by Internet Explorer.

TABLE 12.3 <embed> Attributes

Attribute	Description
allowfullscreen	Specifies whether the embedded element can occupy the full screen. Values are true or false.
allowscriptaccess	Determines whether the embedded object can communicate with external scripts or link to external pages. Values are always, samedomain, and never.
flashvars	Used to pass configuration parameters to the Flash player. Only used if the embedded object is Flash.
height	The height of the element.
plug-inspage	The URL of the page where you can download the plug-in used to view this object.
src	The URL of the multimedia file.
type	The MIME type of the multimedia file indicated by the src attribute.
width	The width of the element.

Finally, you can include the noembed element to provide support for visitors who do not have a web browser that can display plug-ins:

```
<noembed>This Web page requires a browser that can display objects.</noembed>
<embed src="a01607av.avi" height="120" width="160" />
```

The bottom line on <embed> is that you shouldn't use it. I've included it here because you'll probably see it on other sites, but there are better ways to embed media into a web page.

Embedding Flash Movies Using SWFObject

SWFObject is a combination of markup and JavaScript that provides a way to embed Flash movies in web pages using standards-compliant markup that still supports all the browsers that are currently in use. JavaScript is a programming language that runs within the context of a web page. I explain how it works in more detail in Lesson 14, "Introducing JavaScript." You don't need to know how to program in JavaScript to use SWFObject; you just need to copy and paste some code and fill in a few blanks. To download SWFObject and read the documentation, go to http://code.google.com/p/swfobject/.

Aside from providing a reliable way to present Flash movies using standards-compliant markup, SWFObject also works around a problem that can't be dealt with using markup alone. When the version of the Flash player a user has installed is too old to play a movie, the movie will not be presented and any alternative content you provided will not be displayed. The browser hands off the movie to the Flash player assuming it will work and doesn't handle things gracefully if it does not. SWFObject uses JavaScript to catch these errors and show the correct alternative content when they occur.

SWFObject provides two approaches to embedding content, one uses markup augmented by a bit of JavaScript, the other uses pure JavaScript. Using markup provides better performance and offers some level of functionality if JavaScript is disabled or the content is republished in an environment where the JavaScript is not included. The dynamic version is a bit more flexible in that it enables you to configure the embedded player on-the-fly.

Using SWFObject with markup (referred to as static publishing) requires three steps: adding the <object> tags, including the swfobject.js file in the page, and registering the player with the SWFObject library. Here's the code:

```
<script type="text/javascript" src="swfobject.js"></script>
<script type="text/javascript">
  swfobject.registerObject("myId", "9.0.115", "expressInstall.swf");
</script>
<object id="myId" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
width="780" height="420">
  <param name="movie" value="myContent.swf" />
  <object type="application/x-shockwave-flash" data="myContent.swf" width="780"
height="420">
```

```
    <p>Alternative content</p>
  </object>
</object>
```

The first line loads the external JavaScript file into the page. You'll need to make sure that the `src` attribute points to the correct location for your copy of `swfobject.js`. You'll learn more about JavaScript and external JavaScript files in Lesson 14. The next three lines are some JavaScript code that's embedded within the page. Here's the line between the opening and closing `<script>` tag:

```
swfobject.registerObject(object ID, required Flash version, movie URL);
```

The italicized text represents placeholders for the values that you need to plug in to register SWFObject. As you can see, SWFObject requires the ID of the `<object>` tag, the version of Flash that your movie requires, and the URL of the movie to be played.

The other option is to use the JavaScript-based approach. Here's the code:

```
<script type="text/javascript" src="swfobject.js"></script>
<script type="text/javascript">
  swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0");
</script>
<div id="myContent">
  <p>Alternative content</p>
</div>
```

As you can see, the main difference is that the `<object>` tags are gone entirely. Instead, I've got a `<div>` tag that serves as the container for the Flash movie. The alternative content that will be displayed if the Flash player is not present or does not satisfy the version requirement is placed within the `<div>`. The JavaScript call to dynamically publish Flash movies is a bit different from the one used in the static publishing method:

```
swfobject.embedSWF(movie URL, ID of the target div, width, height, required
  Flash version);
```

Many Flash movies enable configuration through a parameter named `FlashVars`. You can specify them using the `<param>` tag:

```
<param name="FlashVars" value="controls=on" />
```

The configuration variables that are available depend entirely on the Flash movie that you're using. You can also configure the movie through the dynamic publishing approach, but it requires a bit more knowledge of JavaScript. For more information, check out the online documentation for SWFObject after the JavaScript lessons.

In the next section, I talk about some specific Flash video players, both of which can be embedded in a page using SWFObject.

Flash Video Players

You've learned how to embed video in pages with the `<video>` tag and how to embed Flash content in pages. Next I introduce some Flash players that can play the same videos you created for use with the `<video>` tag. These players are useful because they enable anyone who has Flash to view your videos. A number of such video players are available. In this section, I discuss two of them: JW Player and Flowplayer.

JW Player

JW Player is a popular Flash video (and audio) player. It is licensed under a Creative Commons noncommercial license, so it's free to use so long as it's not for a commercial purpose. It also requires you to attribute the work to its creator when you use it; in other words, you have to link back to the JW Player website when you embed the player. There's also a commercial version available that you can use for any purpose without the link to the JW Player website. If your website includes advertising, you must use the commercial version of JW Player.

To download JW Player, go to http://www.longtailvideo.com/players/jw-flv-player.

After you've downloaded the player, you'll need your video file as well as the Flash player itself, `player.swf`. JW Player includes an example file that uses the `<object>` and `<embed>` tags, but you should use a standards-complaint approach. You can use SWFObject to embed JW Player, as shown in Figure 12.12, using the following code:

```
<script type="text/javascript" src="swfobject.js"></script>
<script type="text/javascript">
  swfobject.registerObject("myId", "9", "expressInstall.swf");
</script>
<div>
  <object id="myId" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
width="400" height="315">
    <param name="movie" value="player.swf" />
    <param name="flashvars" value="file=video.mp4" />
    <!–[if !IE]>–>
      <object type="application/x-shockwave-flash" data="player.swf"
        width="400" height="315">
        <param name="flashvars" value="file=video.mp4" />
    <!–<![endif]–>
    <div>
      <!– Alternative content –>
      <p><a href="http://www.adobe.com/go/getflashplayer">Download Flash</a></p>
    </div>
    <!–[if !IE]>–>
      </object>
    <!–<![endif]–>
  </object>
</div>
```

FIGURE 12.12
A video played using JW Player.

The code starts with the `<script>` tags used to include the SWFObject script and register the player. If you look at the line that registers the player, you'll see that I'm registering the `<object>` tag with the ID `myId` and that I'm specifying that version 9 of Flash is required, because that's the first version of Flash that supported MP4 and H.264. Finally, there's a reference to `expressInstall.swf`, a Flash movie that's included with SWFObject that enables users to upgrade their Flash player in place if it's out-of-date.

Both the `<object>` tag for Internet Explorer and the nested `<object>` tag for other browsers refer to the JW Player Flash file, `player.swf`. They also both use the `flashvars` param to point to the video file, `video.mp4`. I've also included some alternative content that points the users to the Flash download site if they don't have Flash installed.

JW Player is highly customizable. A wizard on the JW Player website enables you to generate your own customized embed code at http://www.longtailvideo.com/support/jw-player-setup-wizard. The code it produces uses the dynamic publishing feature from an old version of SWFObject. You can see what the wizard looks like in Figure 12.13. There's a detailed list of the configuration parameters in the documentation at the JW Player website. All the configuration options are specified in the `flashvars` parameter. Here's an example that moves the control bar to the top of the player:

```
<param name="flashvars" value="file=video.flv&amp;controlbar=top" />
```

FIGURE 12.13
JW Player with the control bar moved to the top.

There are two configuration parameters in that `<param>` tag, `file` and `controlbar`. Each is separated from its value by an equals sign, and the two parameters are separated by an encoded ampersand. The `flashvars` is formatted in the same way as a URL query string, the same format used for encoding form parameters when they're sent to the server. For more information about how to format query strings, take a look at the Wikipedia article at http://en.wikipedia.org/wiki/Query_string.

Using Flowplayer

Flowplayer is another popular Flash-based video player. The base version is free and open source and can be used on commercial sites. The only catch is that the base version displays the Flowplayer logo at the end of the video. If you want to get rid of the branding or display your own logo, you can purchase a commercial version of Flowplayer. The price is based on the number of domains on which you want to use the player. You can download it at http://flowplayer.org/.

Flowplayer is used similarly to dynamic publishing with SWFObject. To embed a video in a page using Flowplayer, you must include the custom JavaScript file supplied with Flowplayer, using a `<script>` tag:

```
<script type="text/javascript" src="path/to/the/flowplayer-
3.2.2.min.js"></script>
```

Then you have to add a container to the page in which the video will appear:

```
<a href="myvideo.mp4"
       style="display: block; width: 425px; height: 300px;"
       id="player">Download video</a>
```

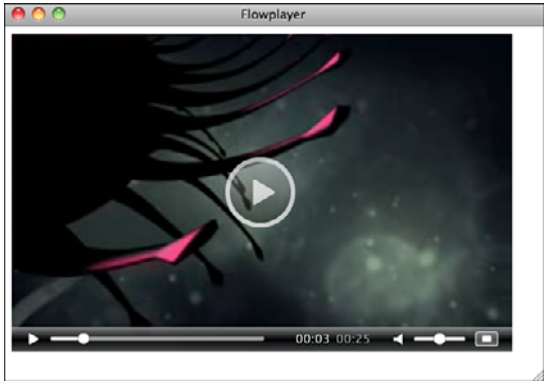
And finally, you need to install the player in the target element:

```
<script type="text/javascript">
flowplayer("player", "path/to/the/flowplayer-3.2.2.swf");
</script>
```

Instead of using the `<object>` tag or using a `<div>` for as the container for the player, Flowplayer recommends using the `<a>` tag. The player will play the video referenced in the `href` attribute of the `<a>` tag. Here's the full example page, which is shown in Figure 12.14:

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="flowplayer-3.2.2.min.js"></script>
</head>
<body>

  <a href="http://e1h13.simplecdn.net/flowplayer/flowplayer.flv"
    style="display:block; width:520px; height:330px"
                        id="player"></a>

  <script type="text/javascript">
    flowplayer("player", "flowplayer-3.2.2.swf");
  </script>
</body>
</html>
```

FIGURE 12.14
A video played using Flowplayer.



There are a number of customization options for Flowplayer. The easiest way to change them is to use Flowplayer's Setup application, available at http://flowplayer.org/setup/. You can also configure Flowplayer yourself using JavaScript. There's a full list of configuration options on the Flowplayer website at http://flowplayer.org/documentation/configuration/.

Using the <object> Tag with the <video> Tag

The <object> tag can be used as an alternative for presenting video for browsers like Internet Explorer that don't support the <video> tag. All that you need to do is included a proper <object> tag inside your <video> tag. The way HTML support works in browsers ensures that this works. Browsers ignore tags that they don't recognize, so Internet Explorer will ignore your <video> tag. Browsers that do support the <video> tag will ignore any <object> tags that are nested within them, recognizing that they are included as an alternative means of presenting the video.

So when you use them together, you wind up with markup that looks like this:

```
<video width="320" height="240" controls>
  <source src="path/to/movie.mp4"
    type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="path/to/movie.ogv"
    type='video/ogg; codecs="theora, vorbis"'>
    <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" ?
      width="320" height="240">
      <param name="movie" value="/movie.mp4"> ?
    </object>
</video>
```

The <video> tag with its <source> elements will present video in browsers that support it, and for those that don't, the <object> element is included to present the video using Flash.

You can even further nest tags, including the <object> tags used by both Internet Explorer and other browsers as children of the <video> tag:

```
<video width="320" height="240" controls>
  <source src="path/to/movie.mp4"
    type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="path/to/movie.ogv"
    type='video/ogg; codecs="theora, vorbis"'>
    <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" ?
      width="320" height="240">
      <param name="movie" value="/movie.mp4"> ?
      <object type="application/x-shockwave-flash" data="myContent.swf"
width="780" height="420">
        <p>You need the Flash player to view this page.
        <a href="http://get.adobe.com/flashplayer/>Get Flash.</a></p>
      </object>
    </object>
</video>
```

In this case, browsers that support the native <video> tag will use it. Then, Internet Explorer will use the nested <object> tag, and finally other browsers will use the inner <object> tag.

Embedding Audio in Your Pages

The nice thing about embedding audio is that it's similar to embedding video. HTML5 provides an `<audio>` tag that works almost identically to the `<video>` tag. The `<embed>` tag can also be used with audio, but you should use the `<audio>` and `<object>` tags instead.

Four main file formats and codecs are used for audio on the Web: MP3, Ogg Vorbis, AAC, and WAV. MP3 is supported natively by Safari and Chrome and can be played using Flash-based players. The WAV format is supported by Firefox, Safari, and Opera. Ogg Vorbis, the open format, is supported by Firefox and Chrome. AAC is the format used by iTunes when you rip CDs. It is supported natively by Safari and also by Flash.

Your best bet for reaching the largest audience is to use the `<audio>` tag with MP3 files for browsers that support it, including mobile browsers that support HTML5 but not Flash, and then use a Flash-based player to play the MP3 files for those users whose browsers do not support HTML5 or don't support the MP3 format.

The `<audio>` Tag

The `<audio>` tag is similar to the `<video>` tag. It attempts to use the native capabilities of the browser to play an audio file. Its attributes are the same as those `<video>` tag, except that the `height` and `width` attributes are not used. Here's an example of the `<audio>` tag:

```
<audio src="song.mp3" controls>
```

If the browser is capable of playing the video at the URL specified in the `src` attribute, it will present the audio, which you can use to control playback. The audio player appears in Figure 12.15.

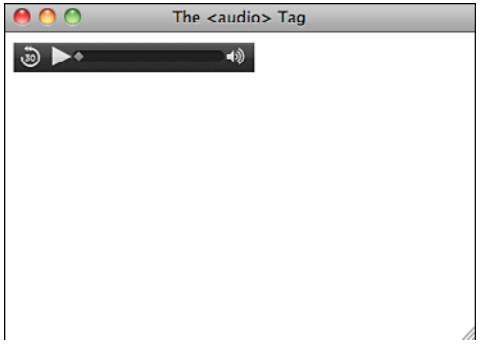


FIGURE 12.15
An embedded audio player.

Table 12.4 lists the attributes of the <audio> tag.

Table 12.4 <audio> Attributes

Attribute	Description
src	The URL for the video to be played.
controls	Boolean attribute that indicates that the browser should supply its own controls for the video. The default is to leave out the controls.
autoplay	Boolean attribute that indicates that the video should play immediately when the page loads.
loop	Boolean attribute. If present, the video will loop when it reaches the end, replaying until the user manually stops the video from playing.
preload	Boolean attribute. If present, the browser will begin downloading the video as soon as the page loads to get it ready to play. Ignored if autoplay is present.

To provide background music for the page, you can add the autoplay and loop attributes to the tag. Chances are, if you use autoplay, or even worse, autoplay and loop without also providing controls for the audio, your users will leave in a hurry.

Flash Audio Players

Embedding Flash audio players is exactly the same as embedding Flash video players in a page. You can use the <object> tag, nested <object> tags, or SWFObject to embed your Flash movie. You can also nest <object> tags within <audio> tags to maximize browser support.

Both JW Player and Flowplayer can play audio as well as video files. To do so, supply the path to an MP3 file rather than to a video file. There are also a number of Flash players that are just for audio.

One popular Flash audio player is just called Audio Player. You'll want to download the standalone version from http://wpaudioplayer.com/. It is also available as a plug-in for the WordPress blogging tool; be sure not to download that version.

After you've downloaded it, copy the audio-player.js and player.swf files to your website. Then, set up the audio player, as shown in this example page, which appears in Figure 12.16:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Audio Player</title>
    <script src="audio-player.js" type="text/javascript"></script>
```

```
    <script type="text/javascript">
      AudioPlayer.setup("player.swf", {
        width: 325
      });
    </script>
  </head>
  <body>
    <p id="myaudio"><a href="song.mp3">Download MP3 file</a></p>
    <script type="text/javascript">
      AudioPlayer.embed("myaudio", {soundFile: "song.mp3"} );
    </script>
  </body>
</html>
```

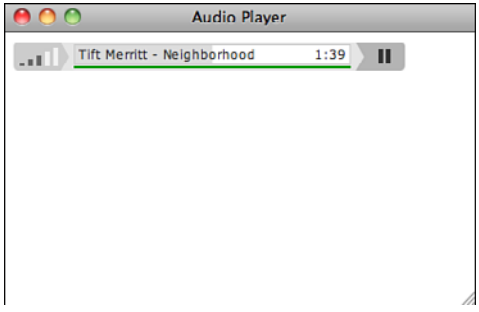


FIGURE 12.16
An embedded audio player.

Audio Player uses JavaScript to embed the player in a target element using a technique similar to the SWFObject dynamic publishing technique. In the header of the page, I included the JavaScript file provided with Audio Player and then set up the player in the next `<script>` tag. Copy that code and replace the size the size that works for you. (The player in my example is set to be 325 pixels wide.)

Then in the body of the page, I included a `<p>` tag that will contain the player. My alternative content goes inside the `<p>` tag. It's replaced when the JavaScript that follows adds the player to the page. In this case, I added a link that points to the MP3 file that the player is going to play. Finally, I embedded the player using JavaScript. For it to work, I specified the ID of the element that will contain the player and specified the location of the audio file. In this case, it's `song.mp3`, found in the same directory as the page.

Summary

In this lesson, you learned about the wide world of tags, codecs, and file formats associated with publishing audio and video on the Web. First, you learned how to upload your video files to YouTube and publish them on your site. You also learned about some

alternative sites, such as Vimeo, that provide video hosting, too. Then you learned about the file formats associated with video on the Web and how to convert videos to those formats. You learned a lot about the limitations of the various browsers in terms of the tags and formats they support and how to work around those limitations to deliver your video and audio content to as many users as possible.

Next, I discussed the option of hosting your own videos. You learned how to embed video in web pages using the `<video>` tag and the `<object>` tag and how to combine them to support the largest number of browsers. You also learned about SWFObject, a tool that makes it easier to embed Flash movies in your pages in a standards-compliant way. The lesson also covered two Flash movies that can used to embed video or audio files in your web pages. Finally, I discussed audio embedding and the `<audio>` tag.

Table 12.5 shows a summary of the tags you learned about in this lesson.

Table 12.5 Tags for Embedding Video and Audio

Tag	Attribute	Use
`<audio>`		Embeds audio files into web pages for native playback by the browser
`<embed>`		Embeds objects into web pages
`<object>...<object>`		Embeds objects into web pages
`<param>...</param>`		Specifies parameters to be passed to the embedded object. Used in the `object` element
`<source>`		Points to a source audio or video file to be played by an `<audio>` or `<video>` tag
`<video>`		Embeds an audio file into a web page for native playback

Workshop

The following workshop includes questions you might ask about embedding video and audio in your web pages, questions to test your knowledge, and three exercises.

Q&A

Q What's the quickest way to get started adding video to my site?

A The quickest way is to use a site like YouTube of Vimeo that makes it easy to upload your video files and then embed them using the code provided. For most publishers, using sites such as these is all that's needed to provide video to users.

Going the extra mile to host your own video is probably not worth it for the vast majority of applications, especially when you can subscribe to a site such as Vimeo for a nominal fee and publish videos hosted there without linking back to them.

Q Should I be worried about web browser compatibility and standards compliance when it comes to audio and video?

A Unfortunately, yes. When it comes to video and audio, it's easy to wind up writing markup that isn't standards-compliant or to leave out part of your audience by using markup that won't work with their browser. Fortunately, as long as you use the techniques listed in this lesson, you can embed video or audio in your pages in a standards-compliant way that supports all the browsers that are currently in use.

Q What is the difference between H.264 and Ogg Theora?

A H.264 and Ogg Theora are both video codecs. They are slightly different in terms of performance, but the main difference is in how they are licensed. Ogg Theora is an open technology that can be implemented by anyone without restraint. H.264 is covered by patents that must be licensed. For this reason, some browsers, like Firefox, do not support H.264 decoding. To use the `<video>` tag and reach the widest number of users, you should make your videos available in both formats.

Quiz

1. How do you accommodate users whose browsers do not support the `<video>` tag and do not have Flash installed?
2. Why is SWFObject a more robust approach to embedding Flash than just using the `<object>` tag?
3. Why are two `<object>` tags required to embed Flash movies in pages that work in most browsers?
4. Which video format is supported by all web browsers that support the `<video>` tag?

Quiz Answers

1. The key to accommodating users who cannot view your video because of the browser they're using or because they don't have the proper plug-in installed is to use alternative content. Content placed inside the `<video>` tag or `<object>` tag will be ignored by browsers that understand those tags and displayed by those that don't. You can include a link to the proper plug-in, a new browser, or even a direct link to the video itself so that the user can download it and play it with an application on their computer.

2. The two main advantages of SWFObject are that it enables you to create valid markup that still supports a wide variety of browsers and that it gracefully handles cases where the user is missing the Flash plug-in or the version of the Flash plug-in they have installed is out of date.
3. Two `<object>` tags are required because one set of attributes works with Internet Explorer and another set of attributes work with other browsers, like Firefox and Safari.
4. This is a trick question. No one container format or codec is supported by all browsers. To reach your entire audience, you must encode your video in multiple formats and use Flash for browsers without native support for the `s` tag.

Exercises

1. Upload a video to YouTube or Vimeo, and then create a web page with that video embedded in it.
2. Use one of the two video players listed in this chapter, JW Player or Flowplayer, to embed a video in a web page. Look into the configuration options and try changing the appearance of the player.
3. Try rewriting the YouTube embed code for a video on the site in a standards-compliant fashion. There's no reason why you must use YouTube's nonstandards-compliant code to embed their movies in your pages.

LESSON 13

Advanced CSS: Page Layout in CSS

One of the major draws of modern CSS is the freedom to replace clunky HTML with structured HTML markup, styled by CSS rules. In Lesson 8, "Using CSS to Style a Site," you learned how to place individual portions of your web page in specific locations using absolute positioning or floating content. You can use the same types of style rules to build the visual structure of the page.

In this lesson, you'll learn about the following:

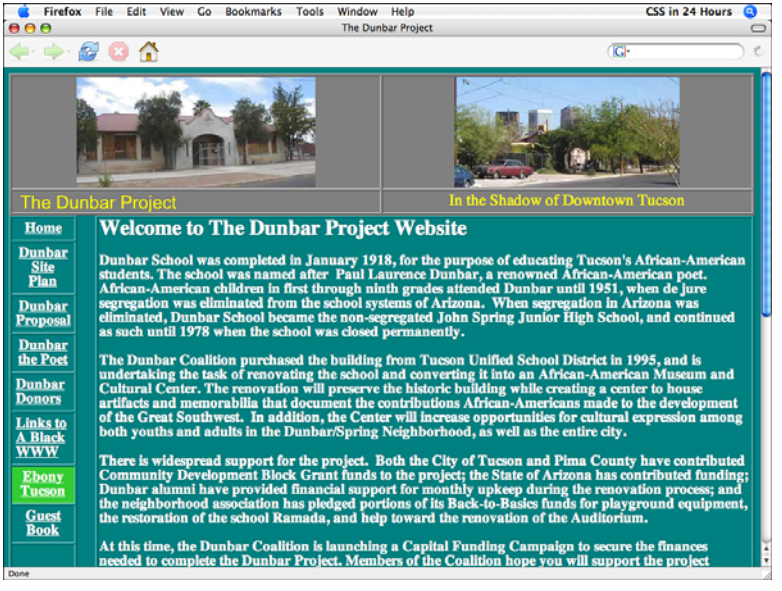
- The different strategies for laying out a page in CSS
- Why it's a bad idea to use `<table>` for page layout
- The steps to replacing a table-based layout with a CSS-based structure
- How to write HTML code for CSS-based layouts
- How to use positioned content to lay out a page
- How to use floating columns to lay out a page
- Which questions you need to ask yourself before starting on a style sheet
- How to organize your style sheets to make them easier to use and edit

Laying Out the Page

This lesson brings together many of the techniques you've learned in previous lessons for using Cascading Style Sheets (CSS) properties to lay out an entire page or even an entire website. You won't need to misuse HTML `<table>` elements for page layout now that you have reliable CSS techniques for layout in your repertoire.

The examples this lesson use a redesigned version of the website for the Dunbar Project in Tucson, Arizona. The site as it appeared before the makeover is shown in Figure 13.1. It is mostly a dark teal color, and although it's not bad, it could be improved through the use of CSS, as you'll see.

FIGURE 13.1
The Dunbar Project's original website.



The Problems with Layout Tables

Figure 13.2 shows the source view for the original version of the Dunbar Project website, which was not designed with CSS. Instead, multiple nested `<table>` tags provide the page layout, and `<font>` is used extensively.

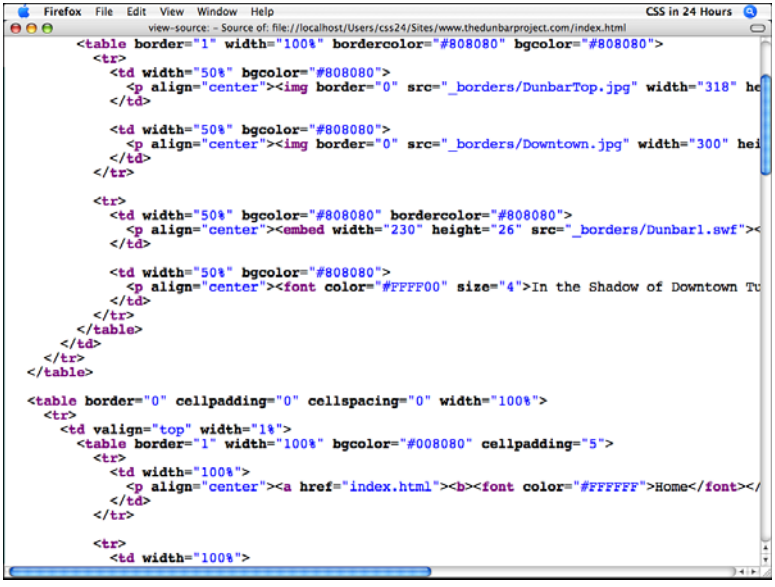


FIGURE 13.2
Table-based layout can be very convoluted.

Tables for layout are problematic for a number of reasons. HTML purists argue against tables on principle: The `<table>` tag is meant to identify data in rows and columns of information and is not intended for page layout. Accessibility mavens will tell you that screen readers employed by visually impaired users struggle with table layout.

Table-based layouts are often harder to maintain than CSS-based layouts, requiring extensive rewriting of HTML tags to make simple changes. Later this lesson, you'll see how a few CSS rules can easily move entire sections around without touching the HTML document at all.

CSS-based layouts make it easier to maintain your HTML pages without cluttering them up with `<tr>` and `<td>` tags and make for simpler transitions to new layouts by just swapping in a new style sheet. Your web pages laid out with CSS will be smaller (and thus load more quickly) than table-based pages. You can write web pages with the main content first in the HTML source and the navigation and footer information after, making your page more friendly to screen readers and search engines.

Writing HTML with Structure

The first step to laying out a page is to start with well-written HTML that is divided into sections for styling. This is commonly done with `<div>` tags that have `id` attributes set on them, corresponding to the different sections of the page.

In Listing 13.1, you can see a redesign of the Dunbar Project home page, which uses simple markup to store the site navigation, the content, the side navigation links, and the page footer.

LISTING 13.1 Using Tags to Create Sections for Positioning

```
<!-- dunbar-13.1.html -->
<html>
  <head>
    <title>The Dunbar Project</title>
  </head>
  <body>
    <div id="header">
      <h1>The Dunbar Project</h1>
      <h2>In the Shadow of Downtown Tucson</h2>
      <div id="sitenav">
        <ol><li><a href="index.html">Home</a></li>
            <li><a href="about/">About the Dunbar Project</a></li>
            <li><a href="gallery/">Photo Galleries</a></li>
            <li><a href="donate/">Donate</a></li>
            <li><a href="contact/">Contact</a></li></ol>
      </div> <!-- sitenav -->
    </div> <!-- header -->
    <div id="main">
      <div id="content">
        <h3>Welcome to The Dunbar Project Website</h3>
        <img src="DunbarTop.jpg" alt="[Dunbar School]">
        <p>Dunbar School was completed in January 1918, for the
           purpose of educating Tucson's African-American students.
           The school was named after <a href="poet.html">Paul
           Laurence Dunbar</a>, a renowned African-American poet.
           African-American children in first through ninth grades
           attended Dunbar until 1951, when de jure segregation was
           eliminated from the school systems of Arizona.  When
           segregation in Arizona was eliminated, Dunbar School
           became the non-segregated John Spring Junior High School,
           and continued as such until 1978 when the school was
           closed permanently.</p>
        <!-- ... more content omitted ... -->
      </div> <!-- content -->
      <div id="sidebar">
        <h3>Dunbar Project</h3>
        <ol><li><a href="plan/">The Dunbar Site Plan</a></li>
            <li><a href="auditorium/">Dunbar Auditorium</a></li>
            <li><a href="history/">School History</a></li>
            <li><a href="proposal/">Project Proposal</a></li>
            <li><a href="donors/">Dunbar Donors</a></li>
            <li><a href="poet.html">About Paul Laurence Dunbar,
                   Poet</a></li>
```

```
            <li><a href="links/">Related Links</a></li></ol>
        <h3>Coalition Partners</h3>
        <ol><li>The Tucson Urban League</li>
            <li>The Dunbar Alumni Association</li>
            <li>The Dunbar/Spring Neighborhood Association</li>
            <li>The Juneteenth Festival Committee</li></ol>
        <h3>Individual Members</h3>
        <ol> <!-- ... list of donors omitted ... --> </ol>
      </div> <!-- sidebar -->
      <div id="footer">
        <p id="note501c3">The Dunbar Project is a 501c(3) organization,
            and your contributions are tax deductible.</p>
        <p id="copyright">Copyright &copy; 2006 by the Dunbar
            Project. Questions?
            <a href="mailto:webmaster@thedunbarproject.com">
            Mail the Webmaster.</a></p>
      </div> <!-- footer -->
    </div> <!-- main -->
  </body>
</html>
```

The structure of this page is defined by the `<div>` tags with `id` attributes. The general skeleton (with content omitted) consists of the following:

```
<div id="header">
  <div id="sitenav"></div>
</div> <!-- header -->
<div id="main">
  <div id="content"></div>
  <div id="sidebar"></div>
  <div id="footer"></div>
</div> <!-- main -->
```

Comments are used with the closing `</div>` tags as reminders about which `<div>` is being closed; it makes the page easier to edit later.

The page is constructed of two sections: a header and a main body. Each of these has one or more subsections. This structure provides what's needed to redesign and lay out the page.

Why this particular structure? There are actually many ways you could structure such the page, inserting `<div>` tags appropriately. This skeleton is simply the method chosen for this example, to get the specific styles used later on. During the web development process, you might go back to your HTML and add or remove `<div>` tags while styling to give more flexibility when creating page layouts.

The number of tags you use will vary from layout to layout. Some web designers believe strongly in using only a minimum number of tags, whereas others add them freely whenever needed. The approach for this example is down the middle between those extremes: There are enough to make it easy to illustrate how CSS-based layout works but not so many that we're adding extraneous tags just because we can.

Figure 13.3 shows the new HTML page without any styles applied.

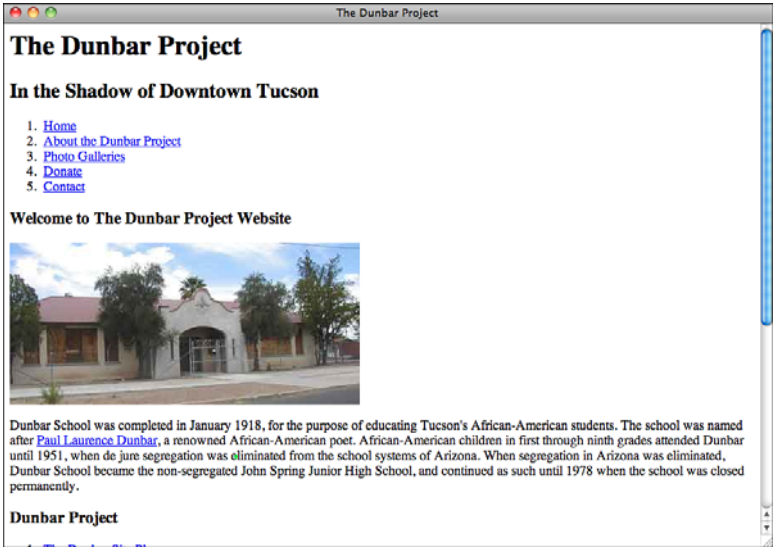


FIGURE 13.3
An unstyled page, ready for layout.

Writing a Layout Style Sheet

With an HTML page ready for styling, the next step is to write the style sheet. There are several questions to consider regarding how to lay out the page.

The first is a technical question: Will you use absolute positioning for layout, or will you use floated columns? You can get the same general layout effects from both techniques. Positioning is a little bit easier to grasp, at first, so this example uses absolute positioning. Later this lesson, however, you'll learn how to lay out the same HTML page with the `float` property.

You need to figure out how many columns you want. There's a slight increase in complexity when you have more columns, but the specific techniques remain the same whether you're using two columns, three columns, or more. In this redesign, two columns are used to avoid making the example overly complex.

Finally, you need to determine whether you are using a fixed layout or a liquid layout. A fixed layout is one that defines a specific width for an entire page; for example, it may be always 700 pixels across, and excess space in the browser simply becomes wider margins. A liquid layout is one that grows larger (or smaller) based on the user's screen resolution and browser window size.

There are advantages and disadvantages to both fixed and liquid layouts. A fixed layout may be easier to create and easier to read on larger monitors; a liquid layout is more adaptable but could result in overly long lines, which are harder to read. In this example, the Dunbar Project site will use a liquid design with margin size based on em units.

Listing 13.2 is a style sheet that starts to set up the layout choices.

LISTING 13.2 A Style Sheet for Page Layout

```
/* dunbar-layout-13.2.css */

body        { margin: 0; padding: 0;
              background-color: silver; }
#header     { background-color: black; color: white; }
#sitenav ol { padding: 0; margin: 0;
              display: inline; }
#sitenav li { display: inline; padding-left: 1em;
              margin-left: 1em; border-left: 1px
              solid black; }
#sitenav li:first-child
            { padding-left: 0; border-left: none;
              margin-left: 0; }
#sitenav li a { color: white; }
#main       { padding: 0 12em 2em 2em;
              position: relative;
              background-color: gray; }
#content    { background-color: white; }
#sidebar    { position: absolute; width: 10em;
              right: 1em; top: 1em; }
#sidebar h3 { color: white;
              background-color: black; }
#sidebar ol { margin: 0 0 1em 0;
              background-color: white;
              border: 2px solid black; }
#footer     { background-color: white; }
```

This style sheet is deliberately plain and simple, with colors of black, gray, silver, and white to make it easier for you to identify the various sections of the page.

So what's happening here?

- The first rule sets the margin and padding of the `<body>` to `0`. This is an important first rule for layout because browsers typically add one or the other (or both) to any web page.
- The `#sitenav` rules in Listing 13.2 are used to turn the ordered list of links into a horizontal navigation bar.
- The `#main` section is set to `position: relative` to become the containing block around the `#content`, `#sidebar`, and `#footer` sections.
- The `#main` section is also given a large padding on the right, `12em`. This is where the `#sidebar` will be located.
- Absolute positioning is used to move the `#sidebar` into the margin, out of its place in the normal flow of content. It is positioned 1 em to the left of the right edge of its containing block (`#main`) by `right: 1em`, and 1 em down from the top edge of the containing block by `top: 1em`.

Figure 13.4 shows the results of linking this style sheet to the HTML file from Listing 13.1

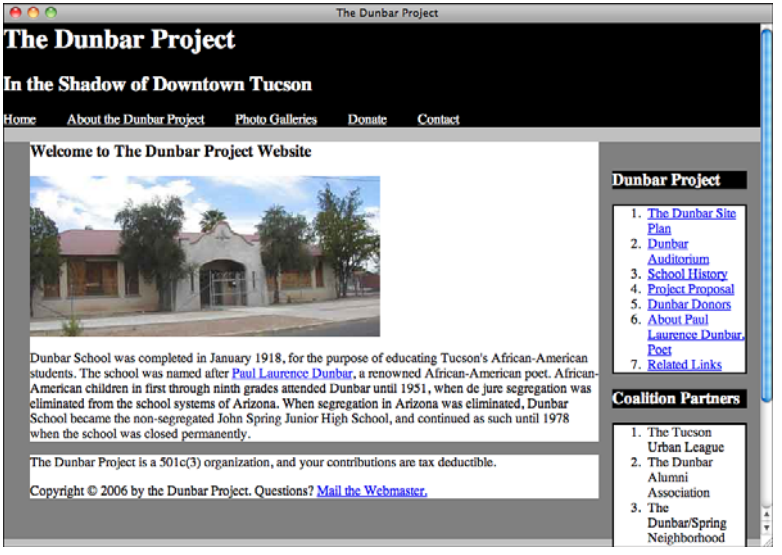


FIGURE 13.4
Positioning properties define the rough outline of the page.

It's still quite rough, but you can see the different sections moved into place. You should note the silver bars above and below the header. Where did they come from, and why?

The silver bars are the result of the background color set on the <body> showing through. They are formed because of the default margin properties set on the <h1> and <h3> headings used on the page. Remember that margins are outside of the border of an element's box, and the background-color property on a box colors only the interior content, not the margin. This applies even when you have a wrapped around a heading, such as <h1>. The margin extends beyond the edge of the 's background-color.

To fix this, we explicitly set the heading margins to zero on the heading tags. Listing 13.3 is a style sheet that not only does that, but also assigns colors, fonts, and other styles on the site. The teal, purple, white, and yellow colors were chosen to reflect not only the original design of the website, but also the actual colors used at the Dunbar school auditorium, too.

LISTING 13.3 A Style Sheet for Colors and Fonts

```
/* dunbar-colors-13.3.css */

body        { font-family: Optima, sans-serif; }
a:link      { color: #055; }
a:visited   { color: #404; }

#header     { text-align: center;
              color: white; background-color: #055; }
#header h1, #header h2
            { margin: 0; }
#header h1  { color: #FFFF00; font-size: 250%; }
#header h2  { font-weight: normal; font-style: italic; }

#sitenav    { color: white; background-color: #404; }
#sitenav ol { font-size: 90%; text-align: center; }
#sitenav li { margin-left: 1em;
              border-left: 1px solid #DD0; }
#sitenav li a:link, #sitenav li a:visited
            { color: white; text-decoration: none; }
#sitenav li a:hover
            { color: #DDDD00; }

#main       { background-color: #055; }

#content    { background-color: white; padding: 1em 5em; }
#content h3 { margin-top: 0; }
#content p  { font-size: 90%; line-height: 1.4; }

#sidebar h3 { font-size: 100%; color: white; margin: 0;
              font-weight: normal; padding: 0.125em 0.25em;
              background-color: #404; }
```

LISTING 13.3 Continued

```
#sidebar ol { background-color: white; border: 2px solid #404;
              border-top: 0; margin: 0 0 1em 0;
              padding: 0.125em 0.25em; }
#sidebar li { font-size: 85%;
              display: block; padding: 0.125em 0; }
#sidebar li a:link, #sidebar li a:visited
            { text-decoration: none; color: #055; }
#sidebar li a:hover { color: #404; }

#footer     { background-color: #404; color: white;
              padding: 0.5em 5em; }
#footer p   { margin: 0em; font-size: 85%; }
#footer p a:link, #footer p a:visited
            { color: #DDDD00; }
```

Figure 13.5 shows the HTML file from Listing 13.1 with both the layout style sheet from Listing 13.2 and the colors and fonts style sheet from Listing 13.3.

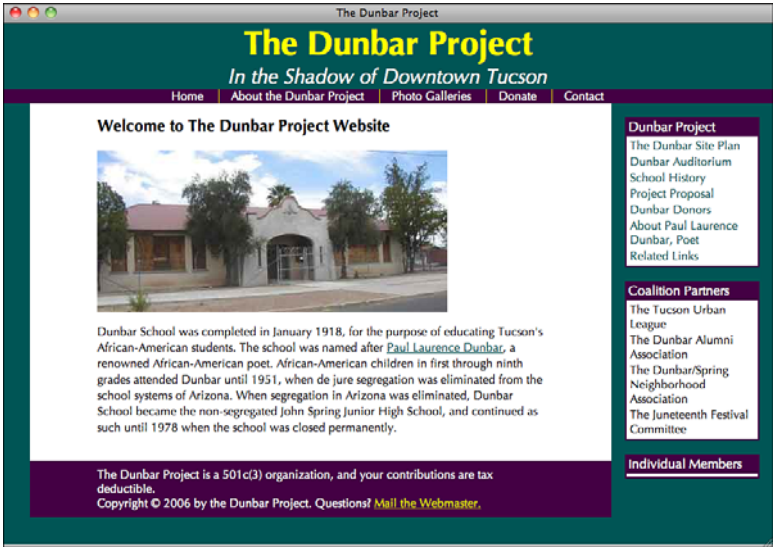


FIGURE 13.5
Fonts and colors help define the website's look.

As you can see, the styled page in Figure 13.5 looks quite different from the unstyled version in Figure 13.3.

Reordering Sections with Positioning Styles

The page in Figure 13.5 looks okay, but let's say that you got this far into the web design process and you suddenly decide that you want to have the site navigation bar located on top of the headline, rather than below it.

You could go in and change your HTML source around. This would work, but it would introduce a problem. The order of the HTML in Listing 13.1 is sensible—the name of the site is given first, and then the navigation menu. This is how users of non-CSS browsers such as Lynx will read your page, and also how search engines and screen readers will understand it as well. Moving the title of the page after the list of links doesn't make much sense.

Instead, you can use CSS positioning properties to reformat the page without touching the HTML file. Listing 13.4 is a style sheet to do exactly that.

LISTING 13.4 Moving One Section Before Another

```
/* dunbar-move-13.4.css */

#header   { padding: 1.25em 0 0.25em 0;
            position: relative;
            background-color: #404; }
#sitenav  { position: absolute;
            top: 0; right: 0;
            border-bottom: 1px solid #DDDD00;
            width: 100%;
            background-color: #055; }
```

What's happening here?

- The `#header` section encloses the `#sitenav` in the HTML source, so by setting it to `position: relative`, it now becomes the containing block for the site navigation links.
- Padding is added to the top of the `#header` section. This is where subsequent rules will place the site navigation menu; the padding reserves the space for it.
- Absolute positioning properties align the top-right corner of the `#sitenav` section with the top-right corner of its containing block, the `#header`.
- Giving a `width` of `100%` to the `#sitenav` ensures it will reach across the full width of its containing block, which is, in this case, as wide as the browser display window.
- Finally, colors are swapped on the `#header` and the `#sitenav` to make them fit in better with the overall design in their new locations, and a yellow border is added to the bottom of the navigation links.

Figure 13.6 shows the effects of these changes.

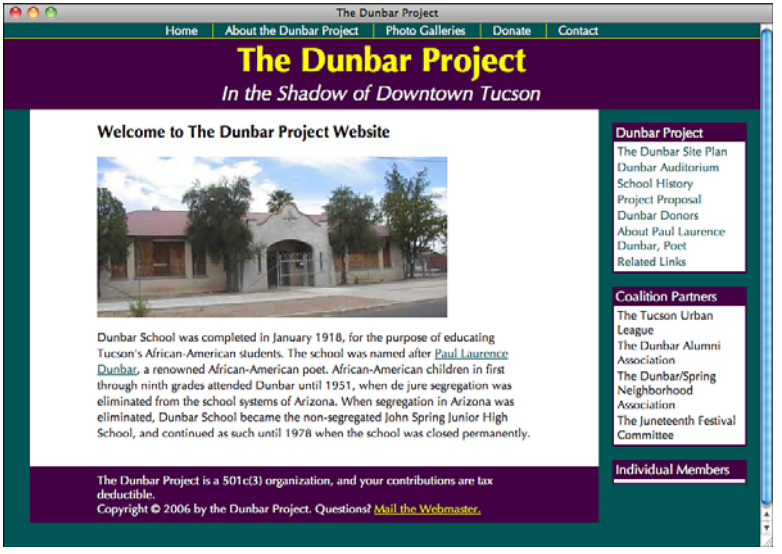


FIGURE 13.6
The navigation menu is now above the page headline.

Task: Exercise 13.1: Redesign the Layout of a Page

You just learned how to move the site navigation menu around. What if you want to make further changes to the page? Try these steps to get familiar with how easy it is to change the layout with CSS:

1. Download a copy of the source code for editing. The file `dunbar.html` contains the complete HTML page, and `dunbar-full.css` has all the style rules listed in this chapter combined into a single style sheet.
2. Move the sidebar to the left side of the page instead of the right. To do this, you need to make space for it in the left gutter by changing the padding rule on the `#main` section to
   ```
   #main { padding: 0 2em 2em 12em; }
   ```
3. Then change the positioning offset properties the `#sidebar`. You don't even have to change the rule for the `top` property; just replace the property name `right` with `left`.
4. Reload the page. You should now see the menu bar on the left side of the screen.
5. Next, move the `#footer` section. Even though the id of the `<div>` is `"footer"`, there's nothing magical about that name that means it needs to be at the bottom of the page. Place it on the right side, where the sidebar used to be located. First clear some space:
   ```
   #main { padding: 0 12em 2em 12em; }
   ```

6. Then reposition the footer with these rules:

```
#footer { position: absolute;
          top: 1em; right: 1em;
          width: 10em;
          padding: 0; }
#footer p { padding: 0.5em; }
```

7. Reload the page. The `#footer` is now no longer a footer, but a third column on the right side of the page.

The Floated Columns Layout Technique

You can also lay out a web page by using the `float` property rather than positioning properties. This method is a little bit more complex but is favored by some designers who prefer the versatility. In addition, floated columns can be written with fewer `<div>` tags and in some cases deal better with side columns that are shorter than the main text.

Listing 13.5 is a style sheet demonstrating how you can float entire columns on a page with CSS. This is a replacement for the `dunbar-layout-13.2.css` style sheet in Listing 13.2. The new style sheet places the menu bar on the left instead of the right, just for variety's sake—there's nothing inherently left-biased about floated columns (or right-biased about positioning).

LISTING 13.5 Float-Based Layouts in CSS

```
/* dunbar-float-13.5.css */

body        { margin: 0; padding: 0; }
#sitenav ol { padding: 0; margin: 0;
              display: inline; }
#sitenav li { display: inline; padding-left: 1em;
              margin-left: 1em; border-left: 1px
              solid black; }
#sitenav li:first-child
            { padding-left: 0; border-left: none;
              margin-left: 0; }

/* This is what positions the sidebar: */
#main       { padding: 0 2em 2em 12em; }
#content    { float: left; }
#sidebar    { float: left; width: 10em;
              position: relative;
              right: 11em; top: 1em;
              margin-left: -100%; }
#sidebar ol { margin: 0 0 1em 0; }
```

What does this style sheet do?

- The first section just duplicates the site navigation bar code from Listing 13.2 so that the entire style sheet can be replaced by this one.
- Starting at the second comment, the code for positioning the columns appears. The first rule sets the `#main` section to have a wide gutter on the left, which is where we will be placing the sidebar.
- Both the `#content` and `#sidebar` sections are set to float. This means that they line up on the left side of the `#main` section, just inside the padding.
- A width is given to the `#sidebar` of `10em`. The size was chosen because that allows 1 em of space around it, after it is placed inside the 12 em gutter set by the `padding` rule on `#main`.
- A negative margin is set on the left side of the `#sidebar`, which actually makes it overlay the `#content` section. Relative positioning is then used, via the `right` and `top` rules, to push the sidebar into the correct place in the gutter.

Figure 13.7 shows this style sheet applied to the HTML file in Listing 13.1, along with the colors and fonts style sheet in Listing 13.3 and the style sheet from Listing 13.4, which relocated the site navigation menu.

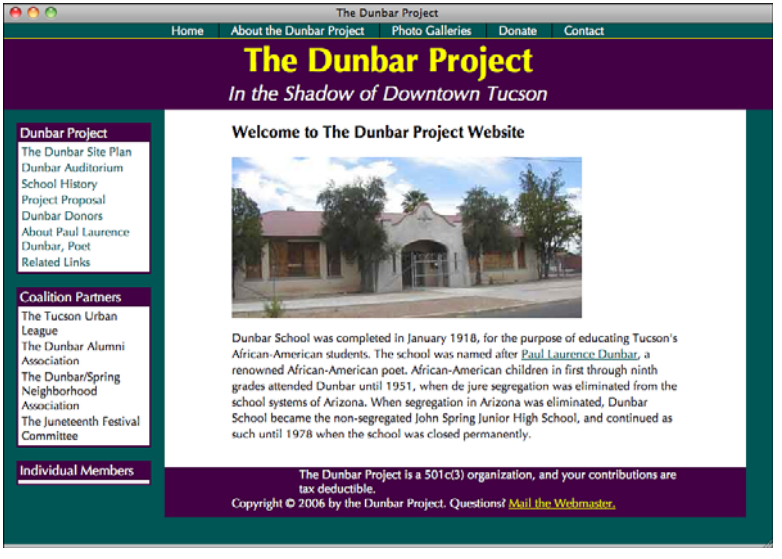


FIGURE 13.7
The sidebar is positioned as floating content.

The Role of CSS in Web Design

As a web developer, skilled in HTML, CSS, and possibly other web languages and technologies, you have a web development process. Even if you haven't planned it out formally, you've got a method that works for you, whether it's as simple as sitting down and designing whatever strikes your fancy or as complex as working in a multideveloper corporate development system for a large employer.

Adding CSS to your repertoire has made you an even better web developer than before; your skill set has expanded and the types of designs you can create are nearly limitless. The next step is to integrate your CSS skills into your web development process. I'm not going to tell you exactly how you'll do that—people have their own methods—but I'll help you think about how you can go about using CSS in your web designs.

In a few cases, you might develop your style sheets completely separately from your HTML pages. More commonly, you'll use an iterative process, where you make changes to the style sheet, then changes to the HTML page, and then go back to the style sheet for a few more tweaks until you're satisfied with the results. The adaptive nature of style sheets makes it easy to create these kinds of changes, and you may find yourself continuing to perfect your styles even after you post your content on the Web.

NOTE

You might not be starting with a blank slate and an uncreated website when you begin using CSS. Redesigns are common in web development, and you may want to take advantage of a new site design to convert to a CSS-based presentation. It can sometimes be harder, but it's certainly possible to keep the same look and feel of your site when converting it to use CSS. If you're using a content management system (CMS) that automatically generates your website from a database, converting to style sheets may be a snap. CSS is very compatible, on a conceptual level, with the idea of templates as used by content management systems.

As mentioned at the start of this lesson, CSS design involves balancing a number of factors to arrive at the best compromise for your site and its users. Questions will arise as you work with CSS on any site, and you'll need to answer them before you go on. Here are several of these key questions to help you plan your site:

- **Will you use Cascading Style Sheets, and if so, to what effect?** You certainly aren't required to use CSS, even after reading this entire book. You can create websites that are usable, accessible, attractive, and effective without a single CSS property anywhere in evidence. However, using CSS will make your site more flexible and easier to maintain and will give you access to presentation effects you couldn't get through HTML alone.
- **What "flavor" of HTML will you use?** HTML5 has a lot of exciting new features, but if you're concerned about older browsers that don't yet understand HTML5, you may want to stick with XHTML.
- **Which browsers will you support?** By "support," I mean investing the effort to work around the quirks of certain older browsers. There are a number of workarounds for these temperamental browsers, plus ways to exclude certain browsers from viewing styles. But if you are designing just for CSS-enabled browsers, such as recent Firefox, Safari, or Opera versions, those workarounds become less important.
- **Are you using positioning CSS for layout?** It's relatively easy to use CSS for formatting text, controlling fonts, and setting colors. Using it for layout is trickier, especially with inconsistent browser support among some of the older versions.
- **Will you use embedded or linked style sheets?** Here, I'll give you advice: Use linked style sheets whenever you can. Some of the examples in this book may use embedded style sheets, but that's mainly because it's easier to give you one listing than two.

The preceding list isn't exhaustive; you'll encounter more choices to make when designing and using CSS, but you should have learned enough by now to answer them.

Style Sheet Organization

The way you organize your style sheet can affect how easy it is for you to use and maintain your CSS, even if the effects are not evident in the presentation. This becomes even more critical if you're in a situation where someone else may have to use your styles in the future. You may work with an organization where multiple people will be working on the same site, or perhaps when you move on to another job your successor will inherit your style sheets.

To make a great style sheet, be organized and clear in what you're doing, and above all, use comments. Web developers often overlook comments in CSS, but if you have to come back later and try to figure out why you did something, they're invaluable. Comments can also be used to group related styles together into sections.

Reasonable names for `class` and `id` attributes can make your style sheet easier to read; choose names for these important selectors that reflect the functions of the elements. If you can, avoid selectors based solely on appearance characteristics, such as the `boldtext` or `redbox` classes. Instead, try something descriptive of why you've chosen those styles, such as `definition` or `sidebar`. That way, if you change your page styles later, you won't have to rewrite your HTML. There are few things as confusing as a rule like the following:

```
.redbox { color: blue; background-color: white; }
```

In what way is that box red? Well, it probably was red in some prior incarnation of the style rules, but not now.

When you list your rules in your style sheet, do them in a sensible order. Generally speaking, it's best to start with the body rules first and then proceed down from there, but because the cascade order matters only in case of conflict, it's not strictly necessary to mirror the page hierarchy. What's more important is that you locate the rules that apply to a given selector and to discern which styles should be applied.

An example of bad style sheet organization is shown in Listing 13.6. This is part of the style sheet from a high-quality website, but with the rules in a scrambled order. How hard is it for you to figure out what is going on here?

LISTING 13.6 A Randomly Organized Style Sheet

```
#sidebar0 .section, #sidebar1 .section { font-size: smaller;
border: 0px solid lime; text-transform: lowercase;
margin-bottom: 1em; }
gnav a:link, #nav a:visited, #footer a:link, #footer
a:visited { text-decoration: none; color: #CCCCCC; }
#nav .section, #nav .shead, #nav .sitem, #nav h1 { display:
inline; }
#sidebar1 { position: absolute; right: 2em; top: 3em;
width: 9em; } a:link { color: #DD8800; text-decoration: none; }
#main { } a:hover { color: lime; }
#nav .shead, #nav .sitem { padding-left: 1em; padding-right:
1em; }
#nav { position: fixed; top: 0px; left: 0px; padding-top:
3px; padding-bottom: 3px; background-color: #333333; color:
white; width: 100%; text-align: center; text-transform:
lowercase; }
#nav .section { font-size: 90%; } #layout { padding: 1em; }
body { background-color: white; color: #333333; font-family:
Verdana, sans-serif; margin: 0; padding: 0; }
#nav h1 { font-size: 1em; background-color: #333333; color:
white; } a:visited { color: #CC8866; text-decoration: none; }
```

LISTING 13.6 Continued

```
#nav { border-bottom: 1px solid lime; } #main { margin-left:
11.5em; margin-right: 11.5em; border: 0px solid lime;
margin-bottom: 1.5em; margin-top: 1.5em; }
#nav a:hover, #footer a:hover { color: lime; }
#sidebar0 { position: absolute; left: 2em; top: 3em;
width: 9em; text-align: right; }
```

If that was hard to follow, don't feel bad; the difficulty was intentional. CSS rules are easily obfuscated if you're not careful. Most style sheets grow organically as piecemeal additions are made; discipline is necessary to keep the style sheet readable.

The style sheet in Listing 13.7 is really the same style sheet as in Listing 13.6. Both are valid style sheets and both produce the same results when applied to the web page, but the second one is easier to understand. Comments make clearer what each section of the style sheet does, indentation and whitespace are used effectively, and the order is much easier to follow.

LISTING 13.7 A Better-Organized Style Sheet

```
/* default styles for the page */
body       { background-color: white;
             color: #333333;
             font-family: Verdana, sans-serif;
             margin: 0;
             padding: 0; }

a:link    { color: #DD8800; text-decoration: none; }
a:visited { color: #CC8866; text-decoration: none; }
a:hover   { color: lime; }

/* layout superstructure */
#layout  { padding: 1em; }

/* top navigation bar */
#nav     { position: fixed;
           top: 0px;          left: 0px;
           color: white;      width: 100%;
           padding-top: 3px;  padding-bottom: 3px;
           background-color: #333333;
           text-align: center;
           text-transform: lowercase; }
           border-bottom: 1px solid lime; }
#nav .section, #nav .shead, #nav .sitem, #nav h1
         { display: inline; }
#nav .section
```

LISTING 13.7 Continued

```
        { font-size: 90%; }
#nav .shead, #nav .sitem
        { padding-left: 1em; padding-right: 1em; }
#nav h1  { font-size: 1em;
           background-color: #333333; color: white; }
#nav a:hover, #footer a:hover
        { color: lime; }
#nav a:link, #nav a:visited,
#footer a:link, #footer a:visited
        { text-decoration: none; color: #CCCCCC; }

/* main content section */
#main    { margin-left: 11.5em;   margin-right: 11.5em;
           margin-bottom: 1.5em;  margin-top: 1.5em;
           border: 0px solid lime; }

/* two sidebars, absolutely positioned */
#sidebar1 { position: absolute;
            right: 2em; top: 3em; width: 9em; }
#sidebar0 { position: absolute;
            left: 2em; top: 3em; width: 9em;
            text-align: right; }
#sidebar0 .section, #sidebar1 .section
          { font-size: smaller;
            border: 0px solid lime;
            text-transform: lowercase;
            margin-bottom: 1em; }
```

Sitewide Style Sheets

The style sheet given in Listing 13.7 was created to be used on the entire site, not just on one page. Linking to an external style sheet is an easy way for you to apply style sheets over your entire set. You just use the `<link>` tag on every page, with the `href` attribute set to the location of your site-wide style sheet.

A sitewide style sheet can be used to enforce a consistent appearance on the website, even if you have multiple web developers working on different parts of the same site. Additional styles can be added in embedded style sheets or in additional linked CSS files that are created for each department or business unit. For example, each department at a school may use the school's global style sheet for design elements common to the entire site, and individual departmental style sheets for that department's unique color, layout, and font choices.

Summary

Tables have long been used in web design to lay out a web page. However, this misuse of `<table>` markup introduces a plethora of complications, from accessibility concerns to complexity problems. Using CSS for layout can clean up your HTML code and produce flexible designs that can be updated easily to new styles.

Laying out a page with CSS starts with adding sections to the HTML, using `<div>`s with ID selectors. These are then arranged in vertical columns, through the use of either positioning rules or the `float` property. With CSS layouts, it's not difficult to reorder and reshape the page simply by changing the style sheet.

Workshop

The workshop contains a Q&A section, quiz questions, and activities to help reinforce what you've learned in this lesson. If you get stuck, the answers to the quiz can be found after the questions.

Q&A

Q **Is it ever okay to use tables for layout?**

A Never, ever, ever! Well, almost. CSS layouts generally are more efficient and versatile than `<table>`-based code, but if you are careful to test your layout tables in a browser such as Lynx to make sure that the site is usable without tables, you can probably get away with it. Tables aren't awful for laying out a page, and CSS can be tricky when you're dealing with grid-based designs. In general, though, you're better off using CSS whenever you can.

Q **Which are better measurements for layouts, pixels or percentages?**

A Some web designers, especially those from a print background or who have picky clients to please, swear by pixels. With some patience, you can get close to pixel-perfect designs in CSS. Other designers like percentage measurements, which scale with the size of the text window. There's no clear-cut advantage to any approach, however; all have their pros and cons. You can experiment with a variety of measurement types, and don't be afraid to mix and match them sensibly on your site—for example, designating column widths in percentages but also setting pixel-based `min-width` and `max-width` values.

Q Are there problems with using ems for layout?

A Only if you're not careful. The biggest problems result from setting margins, padding, or positioning properties based on em values, and then changing the font size of those values. For example, you might overlook the effects of the `font-size` rule buried in these declarations:

```
#sidebar { right: 1em; top: 1em;
      text-align: right; color: white;
      font-family: Verdana, sans-serif;
      font-size: 50%; }
```

This won't actually be located 1 em in each direction from the corner of its containing block; it will be 0.5 em from the right and 0.5 em from the top. If you are going to change the font size within a section that uses ems for dimensions or placement, set the `font-size` rules on the contents of the box, as done in this chapter's style sheets with `#sidebar h3 { ... }` and `#sidebar ol { ... }` rules. You could also add an extra `<div>` inside the sidebar and set the `font-size` rule on that `<div>`.

Quiz

1. Which property tells the text to start flowing normally again, after a floated column?
2. How do you designate the containing block for an absolutely positioned element?
3. What kind of rules would you write to change an ordered list of navigation links into a horizontal navigation bar?

Quiz Answers

1. The `clear` property can be used after floated columns—for example, if you want a footer to reach across the entire browser window below the floated columns.
2. You set the containing block by changing the `position` property, usually to a value of `relative` (with no offset properties designated).
3. Listing 13.7 has an example of a style sheet with rules to do that, using the `display` property.

Exercises

- What kind of layouts can you create with CSS? Choose your favorite sites—either your own or some you enjoy using—and duplicate their layout styles with CSS. Existing sites make good models for doing your own practice, but keep in mind that unless you get permission, you shouldn't just steal someone else's code. Start with the visual appearance as you see it on the screen, and draw out boxes on paper as guidelines showing you where various columns are located. Use that as your model to write the HTML and CSS for building a similar layout.
- Try both of the techniques described in this lesson—using absolutely positioned content and using floating columns. Start with one version and convert it over to the other. Find a style of page that looks right to you, and the CSS code that you think is easiest to understand, apply, and modify consistently.

LESSON 14
Introducing JavaScript

JavaScript is a scripting language that's used to turn web pages into applications. Like *Cascading Style Sheets (CSS)*, JavaScript can be incorporated into web pages in a number of ways. JavaScript is used to manipulate the contents of a web page or to allow users to interact with web pages without reloading the page.

This is the first of three lessons in a row on JavaScript. In this lesson, I explain how JavaScript works and how to use it in your pages. In the next lesson, "Using JavaScript in Your Pages," I walk you through some real-world examples, and then in the following lesson, "Using JavaScript Libraries," I introduce some third-party libraries that you can use to make your life as a JavaScript programmer much easier. In this lesson, you learn about the basics of JavaScript by exploring the following topics:

- What JavaScript is
- Why you would want to use JavaScript
- The `<script>` tag
- An overview of the JavaScript language
- The browser as a programming environment
- Using JavaScript to handle browser events

Why Would You Want to Use JavaScript?

JavaScript was initially introduced with Netscape Navigator 2.0 in 1996. Prior to the introduction of JavaScript, the browser was an application that presented documents generated by a server or stored in files. With JavaScript, the browser became a platform that could run programs. With JavaScript, these programs are included as part of web pages.

JavaScript is useful because it's deeply integrated with the browser. This integration allows programmers to manipulate various aspects of the browser behavior, as well as objects included on an *Hypertext Markup Language (HTML)* page. JavaScript uses what's referred to as an *event-driven model* of execution. When you embed JavaScript code in a web page, it isn't run until the event it's associated with is triggered.

The types of events that can call JavaScript include loading the page, leaving the page, interacting with a form element in some way, or clicking a link. Plenty of other events are available, too. Many of these events are utilized in what most users would consider to be annoying ways. For example, many sites open an additional window containing an advertisement when you navigate to one of their pages. This is accomplished using JavaScript and the page load event. Other sites open additional windows when you leave them; this is also accomplished using JavaScript triggered by an event. Less annoying applications include validating forms before they are submitted, or displaying extra information on a page when a user clicks a link without requiring a full page refresh.

NOTE

This introduction will by necessity be briskly paced. There are many books written about JavaScript. The goal of these lessons is to introduce you to JavaScript, enable you to get started accomplishing tasks, and hopefully kindle your interest to dig into the language more deeply.

JavaScript enables you to manipulate web pages without sending a request back to the server or to send a request to the server to retrieve information without leaving the page that the user is on. Using these capabilities, you can change the contents of a page, change the style of elements on a page, validate user input before a user submits a form, and modify the behavior of the browser—all by using scripts embedded within your web pages. Let's look at some of the advantages of using JavaScript to enhance your web pages.

Ease of Use

JavaScript is a real programming language and is regularly used to build large, complex applications, including some you've probably seen, like Google Maps. At the same time, compared to many other programming languages, it's easy to get started with JavaScript.

You can add useful behavior to a web page with just a little bit of JavaScript added to the `onclick` attribute of a link or to a `script` tag at the top of an HTML document. And as you'll learn in Lesson 16, "Using JavaScript Libraries," JavaScript libraries make it easy to add functionality to web pages using just a few lines of code. The point is, don't be intimidated by JavaScript. You can start accomplishing things almost immediately.

Increasing Server Efficiency

One of the main advantages of JavaScript is that it can provide user feedback instantly. Instead of requiring users to submit a form to see if their input was valid, you can let them know in real time. Not only can this improve user experience, but it can also make life easier for your server, by preventing unnecessary form processing. In other cases, you can use advanced JavaScript applications along with programs on the server to update parts of a page rather than reloading the entire thing. Suppose that you've created a form that people use to enter their billing details into your online ordering system. When this form is submitted, your server-side script first needs to validate the information provided and make sure that all the appropriate fields have been filled out correctly. It needs to check that a name and address have been entered, that a billing method has been selected, that credit card details have been submitted—and the list goes on.

But what happens when the script on the server discovers that some information is missing? You need to alert the visitors that there are problems with the submission and ask them to edit the details and resubmit the completed form. This process involves sending the form back to the browser, having the visitor resubmit it with the right information, revalidating it, and repeating the process until everything is correct. This process can be resource-intensive on the server side and could potentially discourage users from completing their order.

By adding validation and checking procedures to the web browser with JavaScript, you can reduce the number of transactions because many errors will be caught before forms are ever submitted to the server. And because the web server doesn't need to perform as many validations of its own, fewer server hardware and processor resources are required to process form submissions. The side benefit is that users will find your application more responsive because the trip back to the server isn't required for validation.

Integration with the Browser

JavaScript enables you to manipulate objects on the page such as links, images, and form elements. You can also use JavaScript to control the browser itself by changing the size of the browser window, moving the browser window around the screen, and activating or deactivating elements of the interface. Technologies like Flash can provide an interactive interface, but they are not integrated into the browser in the same way that JavaScript is.

The <script> Tag

The <script> tag is used to include a JavaScript script in a web page in much the same way that the <style> tag is use to add a style sheet to a page. The contents of the <script> tag are expected to be JavaScript source code. There are a couple of other ways to use JavaScript in your pages, too, but the <script> tag is a good place to start.

For the best results across all browsers, you should include the type attribute in the script tag, which specifies the type of content that the tag contains. For JavaScript, use text/javascript. HTML5 uses text/javascript as the default value for the type attribute, but for earlier HTML versions of HTML it was required.

The Structure of a JavaScript Script

When you include any JavaScript code in an HTML document (apart from using the <script> tag), you should also follow a few other conventions:

- HTML standards prior to HTML5 required that the <script> tag be placed between the <head> and </head> tags at the start of your document, not inside the <body> tag. Most of the time, putting your <script> tags inside the <head> tag is the right thing to do, but there are some cases where it makes sense to place them elsewhere, which I'll discuss later.
- Unlike HTML, which uses the <!— comment tag —>, comments inside JavaScript code use the // symbol at the start of a line. Any line of JavaScript code that starts with these characters will be treated as a comment and ignored.

Taking these three points into consideration, here's how the <script> tag is normally used:

```
<html>
<head>
<title>Test script</title>
<script language="JavaScript">
// Your JavaScript code goes here
</script>
</head>
<body>
  Your Web contentgoes here
</body>
</html>
```

The `src` Attribute

Besides the `language` attribute, the `<script>` tag can also include an `src` attribute, which allows a JavaScript script stored in a separate file to be included as part of the current web page. This feature enables you to share JavaScript code across an entire website, just as you can share a single style sheet among many pages.

When used this way, the `<script>` tag takes the following form:

```
<script type="text/javascript" src="http://www.example.com/script.js">
```

The `src` attribute will accept any valid URL, on the same server or on another. Naming your JavaScript files with the extension `.js` is customary. When you're loading external scripts in this way, always place the `<script>` tag inside the `<head>` tag.

The JavaScript Language

When JavaScript is included in a page, the browser executes that JavaScript as soon as it is read. Here's a simple page that includes a script, and the output is included in Figure 14.1:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
        <title>A Simple JavaScript Example</title>
</head>
<body>

<p>Printed before JavaScript is run.</p>

<p>
<script type="text/javascript">
    document.write("Printed by JavaScript.");
</script>
</p>

<p>Printed after JavaScript is run.</p>

</body>
</html>
```

Output ▶

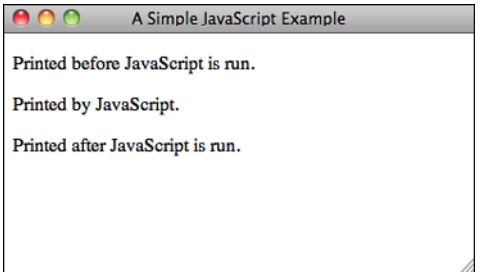


FIGURE 14.1
The results of a simple script.

The page includes a single line of JavaScript, which writes the "Printed by JavaScript" on the page. First of all, that text is printed between the other two paragraphs on the page, demonstrating that the browser executed the JavaScript as soon as it got to it. As you can see, `document.write()` adds text to the page source. What this code does is call the `write` method of the `document` object, which takes a single parameter—the text to be added to the page. For now, it's important to know that the `document` object is a representation of the current page that is accessible by JavaScript. A method represents a type of message that can be sent to an object, the `write()` method tells the document object to add some text to the page. I talk a lot more about the other objects that the browser provides to JavaScript a bit later. Before that, I discuss the argument that I passed to `document.write()` a bit more.

The bit of text that I passed to the `document.write()` method in the previous example is called a string in the vocabulary of programming. When you pass a value to a method, it's called an *argument*. So you say that `document.write()` expects a string argument, which it then adds to the source of the document. JavaScript is a loosely typed language, so even though `document.write()` expects a string, you don't actually have to give a string to work with. You can give it any type of data you want, and it will do its best to turn it into a string and print it on the page. So, for example, you can give it a number:

```
<script type="text/javascript">
       document.write(500);
</script>
```

It will convert the number to a string and print it on the page. Or you can even pass it an object, like this:

```
<script type="text/javascript" charset="utf-8">
       document.write(document);
</script>
```

The results are in Figure 14.2.

Output ▶

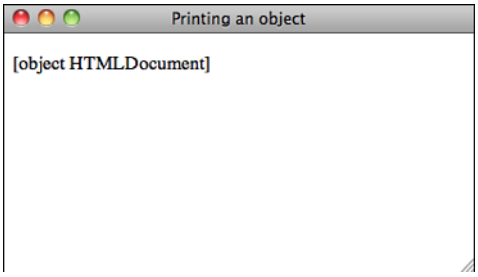


FIGURE 14.2
Attempting to write an object to the page.

That's the string representation of the `document` object. Many programming languages would print an error if you tried to use an object like `document` with a method that accepts a string. Not JavaScript; it makes do with what you give it.

Operators and Expressions

In the previous examples, we supplied a single value to the `document.write()` method as an argument. In the same context, we could have also used an expression. An *expression* is a snippet of code that becomes a value once it has been evaluated. You may recognize the term expression from math, and indeed, many expressions are mathematical expressions. Here's an example of a mathematical expression passed to `document.write()`:

```
<script type="text/javascript">
       document.write(10 * 50);
</script>
```

In this case, JavaScript will multiply 10 by 50 and then pass in the result as the argument. That's an expression. There are also string expressions; you can use the + operator to join strings together, like this:

```
<script type="text/javascript">
       document.write("An" + " " + "expression.");
</script>
```

Expressions are built using operators. You've already seen a couple, * for multiplication and + for combining strings. Table 14.1 lists more operators provided by JavaScript, along with examples. (For a full list of all the supported operators, refer to the online JavaScript documentation.)

TABLE 14.1 JavaScript Operators and Expressions

Operator	Example	Description
+	`5 + 5`	Adds the two numeric values; the result is 10.
+	`"Java" + "Script"`	Combines the two string values; the result is JavaScript.
-	`10 - 5`	Subtracts the second value from the first; the result is 5.
*	`5 * 5`	Multiplies the two values; the result is 25.
/	`25 / 5`	Divides the value on the left by the value on the right; the result is 5.
%	`26 % 5`	Obtains the modulus of 26 when it's divided by 5. (Note: A *modulus* is a function that returns the remainder.) The result is 1.
++	`5++`	Increments the value by 1; the result is 6.
—	`6—`	Decrements the value by 1; the result is 5.

There are many other operators, too. All the examples here used literal values in the expressions, but there are other options, too. You can use values returned by methods in expressions if you choose, as in the following example:

```
<script type="text/javascript">
       document.write(Math.sqrt(25) - Math.sqrt(16));
</script>
```

This example uses the `Math` object, another object built in to JavaScript. It provides a number of methods that perform a variety of mathematical operations so that you don't have to write the code to perform them yourself. `Math.sqrt()` is a method that returns the square root of a number. In this case, I subtracted the square root of 16 from the square root of 25 and passed the result to `document.write()`.

Variables

Thus far, I've been manipulating values and printing them directly to the page. The next fundamental building block of writing scripts is temporary storage of those values so that they can be reused. Like all programming languages, JavaScript supports the use of variables. A *variable* is a user-defined container that can hold a number, text, or an object. Creating a variables and retrieving their values is simple:

```
<script type="text/javascript">
        var message = "My message";
        document.write(message);
</script>
```

In that example, I created a variable called `message` and then passed its value as an argument to `document.write()`. You can also assign the results of an expression to a variable:

```
var sum = 5 + 5;
```

And you can use variables in your expressions:

```
var firstName = "George";
var lastName = "Washington";
var name = firstName + " " + lastName;
```

Let's break down a variable declaration into pieces. Here's a declaration:

```
var message = "My message";
```

The line begins with `var`, which indicates that this is a variable declaration. The name of this variable is `message`. There are a number of rules that apply to naming variables. I list them shortly. The assignment operator (=) is used to assign a value to the variable when it's declared. The value on the right side of the operator is assigned to the newly declared variable.

Variable names must conform to the following rules:

- Variable names can include only letters, numbers, and the underscore (_) or dollar sign ($) character.
- Variable names cannot start with a number.
- You cannot use any reserved words as a variable name. Reserved words are words that have a specific meaning for the JavaScript interpreter. For example, naming a variable named `var` won't work. Table 14.2 contains a full list of JavaScript reserved words.
- As a matter of style, JavaScript variables begin with a lowercase letter. If a variable name contains multiple words, usually an underscore is used to join the two words, or the first letter of the second word is uppercase. So you would write `my_variable` or `myVariable`.

TABLE 14.2 JavaScript Reserved Words

abstract	final	protected
as	finally	public
boolean	float	return
break	for	short
byte	function	static
case	goto	super
catch	if	switch
char	implements	synchronized
class	import	this
continue	in	throw
const	instanceof	throws
debugger	int	transient
default	interface	true
delete	is	try
do	long	typeof
double	namespace	use
else	native	var
enum	new	void
export	null	volatile
extends	package	while
false	private	with

Not all the reserved words in Table 14.2 are currently used in JavaScript some have been placed off limits because they might be added to the language in the future.

Here are a couple of additional notes on variable assignment. You don't have to assign a value to a variable when you declare it. You can declare the variable without an assignment so that it can be used later. For example

```
var myVariable;
```

You can also assign variables after they've been declared, as follows:

```
myVariable = "My value";
```

Finally, the right side of an assignment can be an expression, even an expression that includes another variable:

```
var count = 10;
var sum = 100;
var average = sum/count;
```

Control Structures

To get your scripts to actually do something, you'll need control structures, which come in two main varieties. There are conditional statements, which are used to make decisions, and loops, which enable you to repeat the same statements more than once.

The `if` Statement

The main conditional statement you'll use is the `if` statement. The statements inside an `if` statement are only executed if the condition in the `if` statement is true. If you were writing code in English rather than JavaScript, an `if` statement would read like this: "If the background of this element is blue, turn it red." There's also an `else` clause associated with `if`. The statements in the else clause are executed `if` the if statement's condition is false. An `if` statement with an else clause reads like this: "If the background of this element is blue, turn it red; otherwise, turn it blue."

Let's look at a simple example:

```
var color = "red";
if (color == "blue") {
    color == "red";
} else {
    color == "blue";
}
```

In this example, I've created a variable named `color` and use that in my `if` statement. Later, I explain how to retrieve information from the page, style sheets, and form elements and use them in your JavaScript code. For now, it's easier to explain with hard-coded values. The statement begins with the `if` keyword, followed by the condition enclosed within parentheses. The statements to be executed if the condition is true are placed within curly braces. In this case, I've also included an `else` clause. The statement associated with it is also enclosed in curly braces. Finally, let's look at the condition. It is true if the variable `color` is equal to the value `"blue"`. In this case, the condition is false, so the `else` clause will be executed.

The == operator tests for equality, and is but one of several conditional operators available in JavaScript. Table 14.3 contains all the conditional operators.

Table 14.3 JavaScript Comparison Operators

Operator	Operator Description	Notes
==	Equal to	a == b tests to see whether a equals b.
!=	Not equal to	a != b tests to see whether a does not equal b.
<	Less than less than b.	a < b tests to see whether a is
<=	Less than or equal to	a <= b tests to see whether a is less than or equal to b.
>=	Greater than or equal to	a >= b tests= (greater than or equal to) operator>=)> to see whether a is greater than or equal to b.
>	Greater than	a > b tests (greater than) operator>)> to see whether a is greater than b.

Loops

You'll occasionally want a group of statements to be executed more than once. JavaScript supports two kinds of loops. The first, the `for` loop, is ideal for situations where you want to execute a group of statements a specific number of times. The second, the `while` loop, is useful when you want a set of statements to be executed until a condition is satisfied.

`for` Loops Here's a for loop:

```
for (var count = 1; count <= 10; count++) {
    document.write("Iteration number " + count + "<br />");
}
```

The loop starts with the `for` keyword, followed by all the information needed to specify how many times the loop body will be executed. (Trips through a loop are referred to as *iterations*.) Three expressions are used to define a `for` loop. First, a variable is declared to keep track of the loop iterations. The second is a condition that stops the loop from executing again when it is false. The third is an expression that increments the loop counter so that the loop condition will eventually be satisfied. In the preceding example, I declared the variable `count` with an initial value of `1`. I specified that the loop will execute until the value of `count` is no longer less than or equal to 10. Then I used an operator you haven't seen, `++`, to increment the value of `count` by one each time through the loop.

The body of the loop just prints out a message that includes the value of count each time through the loop.

CAUTION

As you can see, the `for` statement is self-contained. The `count` variable is declared, tested, and incremented within that statement. You shouldn't modify the value of `count` within the body of your loop unless you're absolutely sure of what you're doing.

`while` Loops The basic structure of a `while` loop looks like this:

```
var color = 'blue';
while (color == 'blue') {
    document.write("Color is still blue.");
    if (Math.random() > 0.5) {
        color = 'not blue';
    }
}
```

The `while` loop uses only a condition. The programmer is responsible for creating the condition that will eventually causes the loop to terminate somewhere inside the body of the loop. It might help you to think of a `while` loop as an `if` statement that's executed repeatedly until a condition is satisfied. As long as the `while` expression is true, the statements inside the braces following the `while` loop continue to run forever—or at least until you close your web browser.

In the preceding example, I declare a variable, `color`, and set its value to `"blue"`. The `while` loop will execute until it is no longer true that `color` is set to `"blue"`. Inside the loop, I print a message indicating that the color is still blue, and then I use an `if` statement that may set the `color` variable to a different value. The condition in the `if` statement uses `Math.random()`, which returns a value between `0` and `1`. In this case, if it's greater than 0.5, I switch the value so that the loop terminates.

If you prefer, you can write `while` loops with the condition at the end, which ensures that they always run once. These are called `do ... while` loops, and look like this:

```
var color = "blue";
do {
  // some stuff
}
while (color != "blue");
```

Even though the test in the loop will not pass, it will still run once because the condition is checked after the first time the body of the loop runs.

CAUTION

When you're using `while` loops, avoid creating infinite loops. This means that you must manipulate one of the values in the looping condition within the body of your loop. If you do manage to create an endless loop, about the only option you have is to shut down the web browser. If you're going to iterate a specific number of times using a counter, it's usually best to just use a `for` loop.

Functions

Functions are a means of grouping code together so that it can be called whenever you like. To create a function, you declare it. The following code includes a function declaration:

```
<script language="JavaScript">
function writeParagraph(myString) {
    document.write("<p>" + myString + "</p>");
}
</script>
```

A function declaration consists of the `function` keyword, a function name, a list of parameters the function accepts (in parentheses), and the body of the function (enclosed in curly braces). This function is named `writeParagraph` and accepts a single parameter, `myString`. Function parameters are variables that are accessible within the body of the function. As you can see, this function prints out the value passed in as an argument inside a `<p>` tag. After I've declared this function, I can then use the following code later in the page:

```
<script language="JavaScript">
writeParagraph("This is my paragraph.");
</script>
```

It will produce the output:

```
<p>This is my paragraph.</p>
```

NOTE

When it comes to the values passed to functions, you'll see them referred to as *parameters* or as *arguments*. Technically, the variables listed in the function declaration are parameters, and the values passed to the function when it is called are arguments.

Functions can be written to accept multiple arguments. Let's look at another function:

```
<script language="JavaScript">
function writeTag(tag, contents) {
    document.write("<" + tag + ">" + contents + "</" + tag + ">");
}
</script>
```

This function accepts two arguments, a tag name and the contents of that tag. There's one special statement that's specific to functions, the `return` statement. It is used to specify the return value of the function. You can use the value returned by a function in a conditional statement, assign it to a variable, or pass it to another function. Here's a function with are return value:

```
function addThese(value1, value2) {
    return value1 + value2;
}
```

Here are a couple of examples of how you might use that function:

```
if (addThese(1, 2) > 10) {
    document.write("Sum is greater than 10.");
}
var sum = addThese(1, 2);
```

One other thing to note is that the values passed to function as arguments are copies of those values, unless the values are objects.

Here's one more example, and the results are shown in Figure 14.3:

Input ▼

```
<script language="JavaScript">
function modifyValue(myValue) {
    document.write(myValue + "<br />");
    myValue = "new value";
    document.write(myValue + "<br />");
}

var value = "old value";
modifyValue(value);
document.write(myValue + "<br />");
</script>
```

Output ▶

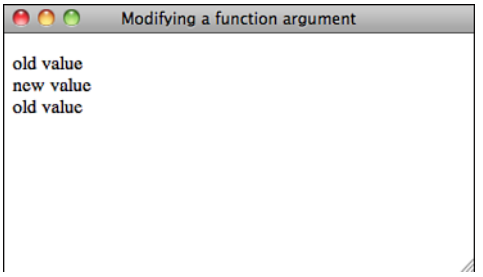


FIGURE 14.3
Values passed to functions are copies.

Functions are called using the function name, followed by parentheses. If you are passing arguments to a function, they are included in the parentheses in a comma-separated list. Even if you're not using arguments, the parentheses are still required. This is true whether you're calling a function you wrote yourself or a function that's built in to JavaScript.

Data Types

I've mentioned JavaScript's type system, but I haven't talked much about JavaScript data types. JavaScript supports the following types of values:

- Strings, like "Teach Yourself Web Publishing".
- Boolean values (true or false).
- Numbers, integer or decimal.
- `null`, which is used to represent an unknown or missing value.
- `undefined`, the value associated with variables that have been declared but have not yet had a value assigned to them.

This is the full set of primitive data types that JavaScript supports. JavaScript attempts to convert data to whatever type it needs in a given context. So if you take a Boolean value and use it in a context where a string is expected, JavaScript will convert it to a string. In some cases, this automatic conversion process can lead to odd results. For example, if you try to use a value that's not a number in a context where a number is expected, JavaScript will return a special value, `NaN`, which is short for "not a number":

```
var squareRoot = Math.sqrt("a string"); // The value of squareRoot is NaN
```

Boolean values represent a state of either true or false. You've already seen some examples that involve boolean values. For example, `if` statements and `while` loops require conditional expressions that return a Boolean value. Any JavaScript value or expression can ultimately be converted to a Boolean. The values that are treated as false are the number zero, empty strings, `null`, `undefined`, and `NaN`. Everything else is true.

Arrays

Arrays are lists of things. They can be lists of values, lists of objects, or even lists of lists. There are a couple of ways to declare arrays. The first is to create your own `Array` object, like this:

```
var list = new Array(10);
```

That declares an array with 10 slots. Arrays are numbered (or indexed) starting at 0, so an array with ten elements has indexes from 0 to 9. You can refer to a specific item in an array by placing the index inside square brackets after the array name. So, to assign the first element in the array, you use the following syntax:

```
list[0] = "Element 1";
```

There are a couple of shortcuts for declaring arrays, too. You can populate an array when you construct the `Array` object like this:

```
var list = new Array("red", "green", "blue");
```

Or you can use what's called an array literal and skip the `Array` object entirely, like this:

```
var list = ["red", "green", "blue"];
```

To find out how many elements are in an array, you can use a property of the array called `length`. Here's an example:

```
listLength = list.length
```

Objects

You've already been introduced to a few objects, most recently, the `Array` object. JavaScript features a number of built-in objects, and the browser supplies even more (which I discuss in the next section). The first thing you need to know about objects is that they have properties. You just saw one property, the `length` property of the `Array` object.

Object properties are accessed through what's known as dot notation. You can also access properties as though they are array indexes. For example, if you have an object named `car` with a property named `color`, you can access that property in two ways:

```
car.color = "blue";
car["color"] = "red";
```

You can also add your own properties to an object. To add a new property to the `car` object, I just have to declare it:

```
car.numberOfDoors = 4;
```

There are a number of ways to create objects, but I just describe one for now. To create an object, you can use an object literal, which is similar to the array literal I just described:

```
var car = { color: "blue", numberOfDoors: 4, interior: "leather" };
```

That defines an object that has three properties. As long as the properties of the object follow the rules for variable names, there's no need to put them in quotation marks. The values require quotation marks if their data type dictates that they do. You can name properties whatever you like, though, as long as you use quotation marks.

In addition to properties, objects can have methods. Methods are just functions associated with the object in question. This may seem a bit odd, but methods are properties of an object that contain a function (as opposed to a string or a number). Here's an example:

```
car.description = function() {
       return color + " car " + " with " + numberOfDoors " and a " + interior +
" interior";
}
```

As you can see, this is a bit different from the function declarations you've seen before. When you declare a method, instead of specifying the function name in the `function` statement, you assign an anonymous function to a property on your object. You can specify parameters for your methods just as you specify them for functions.

After you've added a method to an object, you can call it in the same way the methods of built in objects are called. Here's how it works:

```
document.write(car.description());
```

NOTE

The core JavaScript language contains lot of built-in objects, too many to cover here. For more information about these objects, look at the JavaScript documentation provided by Mozilla or Microsoft.

The JavaScript Environment

I've taken you on a brief tour of the JavaScript language, but beyond the basic language syntax, which involves declarations, control structures, data types, and even core objects that are part of the JavaScript language, there's also the browser environment. When your

scripts run, they have access to the contents of the current page, to other pages that are open, and even to the browser itself. I've mentioned the `document` object, which provides access to the contents of the current page.

Now let's look at a specific object. The top-level object in the browser environment is called `window`. The `window` object's children provide information about the various elements of a web page. Here are some of the most commonly used children of `window`:

`location`	Contains information about the location of the current web document, including the URL and components of the URL such as the protocol, domain name, path, and port.
`history`	Holds a list of all the sites that a web browser has visited during the current session and also gives you access to built-in functions that enable you to send the user forward or back within the history.
`document`	Contains the complete details of the current web page. All the tags and content on the page are included in a hierarchy under document. Not only can you examine the contents of the page by way of the `document` object, but you can also manipulate the page's contents.

You can find a complete list of the available objects in the Mozilla JavaScript documentation at http://developer.mozilla.org/en/JavaScript.

Because the entire browser environment is accessible through this hierarchical set of objects, you can access anything as long as you know where it lives in the hierarchy. For example, all the links on the current page are stored in the property `document.links`, which contains an array. Each of the elements in the array have their own properties as well, so to get the location to which the first link in the document points, you use `document.links[0].href`.

Events

All the examples you've seen so far are executed as soon as the page loads. JavaScript is about making your pages more interactive, which means writing scripts that function based on user input and user activity. To add this interactivity, you need to bind your JavaScript code to events. The JavaScript environment monitors user activity and provides the opportunity for you to specify that code will be executed when certain events occur.

There are two ways to bind JavaScript code to an event handler. The first to do it by way of an attribute on a tag associated with that event handler. The second is to locate the tag you want to bind the event to in the *Document Object Model (DOM)* and then

programmatically associate the code you want to run when that event occurs with the appropriate event handler for that tag. Table 14.4 provides a list of the event handlers that JavaScript provides.

TABLE 14.4 JavaScript Event Handlers

Event Handler	When It's Called
`onblur`	Whenever a visitor leaves a specified form field
`onchange`	Whenever a visitor changes the contents of a specified form field
`onclick`	Whenever a visitor clicks a specified element
`onfocus`	Whenever a visitor enters a specified form field
`onload`	Whenever a page and all its images have finished loading
`onmouseover`	Whenever a visitor places the mouse cursor over a specified object
`onselect`	Whenever a visitor selects the contents of a specified field
`onsubmit`	Whenever a visitor submits a specified form
`onunload`	Whenever the current web page is changed

First, let me explain how to bind an event using HTML attributes. All the event handlers listed above can be used as attributes for tags that respond to them. For example, the `onload` handler is associated with the `body` tag. As you know, JavaScript code is executed as soon as it is encountered. Suppose you want to write a script that modifies all the links on a page. If that script is executed before all the links have been loaded, it will miss some of the links. Fortunately, there's a solution to this problem. The `onload` event does not occur until the entire page has loaded, so you can put the code that modifies the links into a function and then bind it to the page's `onload` event. Here's a page that uses `onload`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Modifying Links with JavaScript</title>

    <script type="text/javascript">
        function linkModifier() {
            for (var i = 0; i < document.links.length; i++) {
                document.links[i].href = "http://example.com";
            }
        }
     </script>
</head>
<body onload="linkModifier()">
```

```
    <ul>
        <li><a href="http://google.com/">Google</a></li>
        <li><a href="http://www.nytimes.com/">New York Times</a></li>
     </ul>
</body>
</html>
```

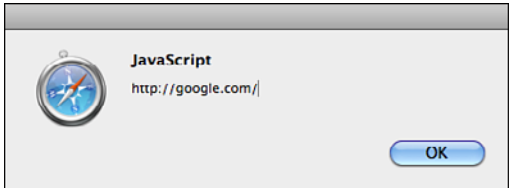
This page contains a script tag in the header, and that script tag contains a single function declaration. The function, `linkModifier()`, changes the `href` attribute of all the links on the page to `http://example.com/`. To access the links, it uses `document.links`, which is a reference to an array of all the links in the document. It iterates over each of those links, changing their `href` properties from the values specified in the HTML to the new URL. The main point of this example, though, is the `onload` attribute in the `body` tag, which contains the call to `linkModifier()`. It's important to associate that call with the `onload` event so that all the links on the page have been processed before the function is called. If I'd put the function call inside the `<script>` tag, the function call might have occurred before the page was loaded.

Most often, when using attributes to bind events, function calls are used, but the value of the attributes can be any JavaScript code, even multiple statements, separated by semicolons. Here's an example that uses the `onclick` attribute on a link:

```
<a href="http://google.com/" onclick="alert(this.href); return
false;">Google</a>
```

In this example, the value of the `onclick` attribute contains two statements. The first uses the build in `alert()` function to display the value in the `href` attribute of the link. The second prevents link from taking the user to a new page. So clicking the link will display the alert message in Figure 14.4 and do nothing after the user acknowledges the alert.

FIGURE 14.4
A JavaScript alert message.



Whether you're writing code in your event binding attribute or writing a function that will be used as an event handler, returning false will prevent the default browser action for that event. In this case, it prevents the browser from following the link. If the `onsubmit` action for a form returns false, the form will not be submitted.

The Meaning of `this`

You might be a bit puzzled by the use of `this` as a variable name in an event handler. Here, `this` is shorthand for the current object. When you're using an event handler in a tag, `this` refers to the object represented by that tag. In the previous example, it refers to the link that the user clicked on. The advantage of using `this` is that it places the event in a useful context. I could use the same attribute value with any link and the code would still work as expected. It's particularly useful when you're using functions as event handlers and you want to make them easy to reuse. You'll see a lot more of `this` in the next lesson.

At one time, using event-handler attributes to bind functions to events was the most common approach, but these days, it's more common to bind events to elements in other ways. It's considered poor style to include JavaScript throughout your web pages, and using the event-handler attributes can override event bindings that are applied from JavaScript rather than in the HTML. In Lesson 15, I explain how to bind events to elements without changing your markup at all.

Learning More About JavaScript

The list of statements, functions, and options included in this lesson represents only part of the potential offered by JavaScript.

For this reason, I cannot overemphasize the importance of the online documentation provided by Netscape. All the latest JavaScript enhancements and features will be documented first at http://developer.mozilla.org/en/JavaScript.

Summary

JavaScript enables HTML publishers to include simple programs or scripts within a web page without having to deal with the many difficulties associated with programming in high-level languages such as Java or C++.

In this lesson, you learned about the `<script>` tag and how it's used to embed JavaScript scripts into an HTML document. In addition, you explored the basic structure of the JavaScript language and how to use JavaScript in the browser environment.

With this basic knowledge behind you, in the next lesson, you'll explore some real-world examples of JavaScript and learn more about the concepts involved in JavaScript programming.

Workshop

The following workshop includes questions, a quiz, and exercises related to JavaScript.

Q&A

Q Don't I need a development environment to work with JavaScript?

A Nope. As with HTML, all you need is a text editor and a browser that supports JavaScript. You might be confusing JavaScript with Java, a more comprehensive programming language that needs at least a compiler for its programs to run. However, tools like FireBug for Firefox, the Internet Explorer Developer Toolbar, and Safari's Web Inspector can make your life easier. Consult the documentation on those tools to learn more about their JavaScript features.

Q What is AJAX?

A One topic we haven't covered yet is AJAX. AJAX is a term used to describe scripts that communicate with the server without requiring a web page to be fully reloaded. For example, you can use it to fetch information and display it on the page, or to submit a form for processing, all without changing the full page in the browser. I discuss AJAX in detail in Lesson 16.

Q When I use JavaScript, do I need to accommodate users whose browsers may not support JavaScript or who have disabled it?

A Some estimates indicate that more than 90% of web users have JavaScript enabled. However, unless you have a really good reason not to, you should make accommodations for users without JavaScript. You need not offer users who don't have JavaScript an identical experience to those who have it, but they should be able to access your site. For example, if you run an online store, do you really want to shut out users because of their browser configuration?

Q In Java and C++, I previously defined variables with statements such as `int`, `char`, and `String`. Why can't I do this in JavaScript?

A As I mentioned previously, JavaScript is a loosely typed language. This means that all variables can take any form and can even be changed on-the-fly. As a result, the context in which the variable is used determines its type.

Quiz

1. What HTML tag is used to embed JavaScript scripts in a page?
2. What are *events*? What can JavaScript do with them?

3. Is an expression that evaluates to the value `0` true or false? How about the string `"false"` inside quotation marks?
4. How do you make sure that a variable you create in a function is only visible locally in that function?
5. How are functions different from methods?

Quiz Answers

1. To accommodate the inclusion of JavaScript programs in a normal HTML document, Netscape introduced the `<script>` tag. By placing a `<script>` tag in a document, you tell the web browser to treat any lines of text inside the tag as script rather than as content for the web page.
2. Events are special actions triggered by things happening in the system (windows opening, pages being loaded, forms being submitted) or by reader input (text being entered, links being followed, check boxes being selected). Using JavaScript, you can perform different operations in response to these events.
3. The number `0` is false, and the string `"false"` is true. The only false values are `0`, `null`, an empty string, `undefined`, `NaN` (not a number), and the Boolean value `false`.
4. The `var` statement is used to define a local variable inside a function.
5. Methods are associated with a specific object, and functions are standalone routines that operate outside the bounds of an object.

Exercises

1. If you haven't done so already, take a few minutes to explore the documentation for JavaScript at https://developer.mozilla.org/en/JavaScript. See whether you can find out what enhancements were included in the latest version of JavaScript that weren't included in earlier versions.
2. Find a simple JavaScript script somewhere on the Web—either in use in a web page or in an archive of scripts. Look at the source code and see whether you can decode its logic and how it works.

LESSON 15

Using JavaScript in Your Pages

Now that you have some understanding of what JavaScript is all about, you're ready to look at some practical applications of JavaScript.

In this lesson, you learn how to complete the following tasks:

- Validate the contents of a form
- Create a list that expands and collapses
- Modify the styles of elements on a page dynamically

Validating Forms with JavaScript

Remember the example form that you created back in Lesson 11, "Designing Forms?" It's shown again in Figure 15.1. It's a typical registration form for a website with several required fields.

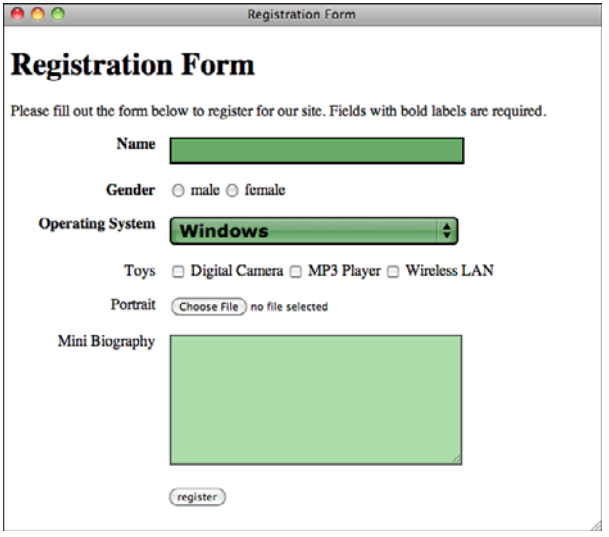


FIGURE 15.1
The registration form.

What happens when this form is submitted? In the real world, a script on the server side validates the data that the visitor entered, stores it in a database or file, and then thanks the visitor for her time.

But what happens if a visitor doesn't fill out the form correctly—for example, she doesn't enter her name or choose a value for gender? The script can check all that information and return an error. But because all this checking has to be done on the server, and because the data and the error messages have to be transmitted back and forth over the network, this process can be slow and takes up valuable resources.

JavaScript enables you to do error checking on the browser side before the form is ever submitted to the server. This saves time for both you and your visitors because everything is corrected on the visitors' side. After the data actually gets to your script, it's more likely to be correct.

Task: Exercise 15.1: Form Validation

Now take a look at how the registration form is validated with JavaScript. Whenever you click the Submit button on a form, two events are triggered: the `onclick` event for the button and the `onsubmit` button for the form. For form validation, `onsubmit` is the better choice, because some forms can be submitted without clicking the Submit button. When the `onsubmit` event is fired, the validation function will be called.

The `onsubmit` attribute is set in the `<form>` tag, like this:

```
<form method="post"
   action="http://www.example.com/register"
   onsubmit="return checkform(this)">
```

In this example, the value assigned to `onsubmit` is a call to a function named `checkform()`—which is defined in the page header. But first, the `return` statement at the beginning of the `onsubmit` field and the `this` argument inside the parentheses in the `checkform()` function need some further explanation.

As you might remember from the preceding lesson, the argument to the `checkform()` function, `this`, has a special meaning. In the context of an event handler, it is a reference to the tag that the event handler is associated with. So in this case, when `checkform()` is called, I pass it a reference to the form, from which I can access all the form fields that I want to validate. The `onsubmit` handler can accept a return value. If the return value is false, then the action is canceled. If it is true, then the form submission continues. So by including the return statement in the event handler, I pass the return value of `checkform()` to the event handler, enabling me to stop the form from being submitted if the values in the form are invalid.

The Validation Function In a `<script>` tag inside the `<head>` block, you will be declaring the `checkform()` function. . The function accepts a single parameter, which you'll name `thisform`. It is a reference to the form that will be validated. The code for the function declaration looks like this:

```
<script language="JavaScript">
function checkform(thisform) {

    // Validation code
    return true;
}
</script>
```

I went ahead and stuck `return true` in the function. If there are no invalid fields, the form submission should be successful.

Testing For Required Fields The form has three required fields: Name, Gender, and Operating System. The validation function is responsible for checking that they each contain a value. The `name` field is a text input field. Here's the markup for the field:

```
<input name="name" type="text" />
```

You use this name to reference the field as a child of `thisform`. The field `name` can be referenced as `thisform.name`, and its contents can be referenced as `thisform.name.value`.

Using this information and an `if` statement, you can test the contents of `name` to see whether a name has been entered:

```
if (!thisform.name.value) {
    alert("You must enter a name.");
    return false;
}
```

The expression in the `if` statement evaluates the value of the form field in a Boolean context. So if the field's value is null or an empty string, the expression will be false. In that case, the function displays an error message and prevents the form from being submitted.

If the name is valid, the next step is to validate the gender radio button group. Validation of these fields differs slightly because radio button groups consist of multiple fields with the same name. So in this case, `thisform.gender` will return an array, and we'll use a loop to iterate over each element in the array and make sure one of the buttons is selected. For radio buttons, the `value` property is the same as the `value` attribute of the form field. The `checked` property, a Boolean value, indicates whether the button is selected.

To test whether one of the `gender` radio buttons has been selected, declare a new variable called `selected` and give it a value of `false`. Then loop through all the radio buttons using a `for` loop; if the `checked` property of any radio button is `true`, set `selected = true`. Finally, if `selected` is still `false` after you've tested each of the radio buttons, display an `alert()` message and exit the function by calling `return false`. The code required to perform these tests is shown here:

```
var selected = false;

for (var i = 0; i < thisform.gender.length; i++) {
    if (thisform.gender[i].checked) {
        selected = true;
    }
}
```

```
    if (!selected) {
        alert("You must choose a gender.");
        return false;
    }
```


The form indicates that the Operating System field is also required, but because it's not possible to not select an item in that select list, there's no need to validate it.

The Completed Registration Form with JavaScript Validation When the JavaScript script that you just created is integrated with the original registration form document from Lesson 11, the result is a web form that tests its contents before they're transmitted to the server for further processing. This way, no data is sent to the server until everything is correct. If a problem occurs, the browser informs the user (see Figure 15.2).

FIGURE 15.2
An alert message.

So that you don't need to skip back to the exercise in Lesson 11 to obtain the *Hypertext Markup Language (HTML)* source used when creating the form, here's the completed form with the full JavaScript code.

```
<!DOCTYPE html>
<html>
<head>
<title>Registration Form</title>
<script language="JavaScript">
    function checkform(thisform) {
        if (!thisform.name.value) {
            alert("You must enter a name.");
            return false;
        }

        var selected = false;

        for (var i = 0; i < thisform.gender.length; i++) {
            if (thisform.gender[i].checked) {
                selected = true;
            }
        }
```

```
        if (!selected) {
            alert("You must choose a gender.");
            return false;
        }

        // Validation code
    return true;
    }
</script>

<style type="text/css" media="screen">
    form div {
        margin-bottom: 1em;
    }

    div.submit input {
        margin-left: 165px;
    }

    label.field {
        display: block;
        float: left;
        margin-right: 15px;
        width: 150px;
        text-align: right;
    }

    input[type="text"], select, textarea {
        width: 300px;
        font: 18px Verdana;
        border: solid 2px #666;
        background-color: #ada;
    }

    div.required label.field {
        font-weight: bold;
    }

    div.required input, div.required select {
        background-color: #6a6;
        border: solid 2px #000;
        font-weight: bold;
    }
</style>
</head>
<body>
<h1>Registration Form</h1>

<p>Please fill out the form below to register for our site. Fields
with bold labels are required.</p>
```

```
<form method="post" onsubmit="return checkform(this)" enctype="multipart/
  form-data">

    <div class="required">
        <label class="field" for="name">Name</label>
        <input name="name" type="text" />
    </div>

    <div class="required">
        <label class="field">Gender</label>
        <label><input type="radio" name="gender" value="male" /> male</label>
        <label><input type="radio" name="gender" value="female" />
female</label>
    </div>

    <div class="required">
        <label class="field">Operating System</label>

        <select name="os">
            <option value="windows">Windows</option>
            <option value="macos">Mac OS</option>
            <option value="linux">Linux</option>
            <option value="other">Other ...</option>
        </select>
    </div>

    <div>
        <label class="field">Toys</label>
        <label><input type="checkbox" name="toy" value="digicam" /> Digital
           Camera</label>
        <label><input type="checkbox" name="toy" value="mp3" /> MP3
           Player</label>
        <label><input type="checkbox" name="toy" value="wlan" /> Wireless
           LAN</label>
    </div>

    <div>
        <label class="field">Portrait</label>
        <input type="file" name="portrait" />
    </div>

    <div>
        <label class="field">Mini Biography</label>
        <textarea name="bio" rows="6" cols="40"></textarea>
    </div>

    <div class="submit">
        <input type="submit" value="register" />
    </div>
</form>
</body>
</html>
```

Improving Form Validation The form validation code in this exercise works, but it could be better. The biggest problem is that it validates only one field at a time—if two fields are invalid, they'll get an error message only for the first one that has a problem. Fortunately, the script can be tweaked to validate all the fields at once. The key is to accumulate the error messages and display them all simultaneously at the end of the function before the return statement.

An array will be used to store the error messages and the return statements, and at the end, if there are elements in the array, the error message will be created and displayed, and the validation routine will prevent the form from being submitted.

Here's the script:

```
<script language="JavaScript">
function checkform(thisform) {
    var errors = [];
    var errorMessage = "Please correct the following errors:\n\n";

    if (!thisform.name.value) {
        errors.push("You must enter a name.");
    }

    var selected = false;

    for (var i = 0; i < thisform.gender.length; i++) {
        if (thisform.gender[i].checked) {
            selected = true;
        }
    }

    if (!selected) {
        errors.push("You must choose a gender.");
    }

    if (errors.length > 0) {
        for (var i = 0; i < errors.length; i++) {
            errorMessage += errors[i] + "\n";
        }

        alert(errorMessage);

        return false;
    }
    else {
        return true;
    }
}
</script>
```

The script starts with the declaration of two variables that will be used in the validation process. The first is an empty array named `errors` that will be used to store any error messages associated with invalid fields. The second is called `errorMessage`, it will store the error message that will be displayed if the user fails to fill out the form correctly. The beginning of the error message is in the declaration and before it is displayed, the specific error messages will be tacked onto it.

If you look at the message, you'll see `\n\n` at the end. Alert messages are not formatted using HTML and do include any whitespace that is inside the string. Each `\n` adds a line feed in the dialog box. Including the line breaks skips a line between that message and the individual error messages.

When my variables are all set up, I start validating the contents of the form. First, I validate the name. If it's empty, I add an error message to the `errors` array using the `push()` method, which adds its argument onto the end of the array. Next, I check the gender field, which also uses the `push()` method to add a message to `errors` if no value has been selected.

After the fields have been validated, it's time to decide whether to display the error message. I test the `length` property of `errors`. If the array contains any elements, it indicates that errors were found. In that case, the error message is displayed and the function returns `false` so that the form is not submitted. If the array is empty, the form is valid and submission can proceed. Figure 15.3 shows the full error message.

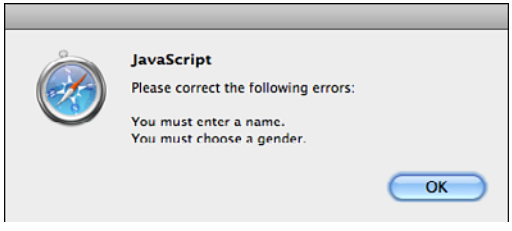


FIGURE 15.3
An alert message for multiple fields.

Hiding and Showing Elements

One way to help users deal with a lot of information presented on a single page is to hide some of it when the page loads and to provide controls to let them view the information that interests them. In this example, I create a frequently asked questions page that displays all the questions by default but hides the answers to the questions. Users can display the answers by clicking the corresponding questions.

In this example, I apply an approach referred to as unobtrusive JavaScript. The philosophy behind it is that the behavior added by JavaScript should be clearly separated from the presentation applied using HTML and *Cascading Style Sheets (CSS)*. The page should work and be presentable without JavaScript and JavaScript code should not be mixed with the markup on the page.

In practice, what this means is that events handlers should be specified in scripts instead of HTML attributes. Also, in this example, using JavaScript unobtrusively means that if the user has JavaScript turned off, they will see all the questions and answers by default, rather than seeing a page with the questions hidden and no means of displaying them.

▼ Task: Exercise 15.2: Hiding and Showing Elements

The exercise starts with a web page that displays the frequently asked questions that doesn't include any JavaScript. Here's the source for the page. Figure 15.4 shows the page in a browser.

Input ▼

```
<!DOCTYPE html>
<html>
<head>
    <title>Frequently Asked Questions</title>
    <style type="text/css" media="screen">
        dt { margin-bottom: .5em; font-size: 125%;}
        dd { margin-bottom: 1em;}
    </style>
</head>
<body>

<h1>Frequently Asked Questions</h1>

<dl id="faq">
    <dt>Don't I need a development environment to work with JavaScript?</dt>
    <dd>Nope. As with HTML, all you need is a text editor and a browser that
supports JavaScript. You might be confusing JavaScript with Java, a more compre-
hensive programming language that needs at least a compiler for its programs to
run. However, tools like FireBug for Firefox, the Internet Explorer Developer
Toolbar, and Safari's Web Inspector can make your life easier. Consult the docu-
mentation on those tools to learn more about their JavaScript features.</dd>

    <dt>What is AJAX?</dt>
    <dd>One topic we haven't covered yet is AJAX. AJAX is a term used to
describe scripts that communicate with the server without requiring a Web page
to be fully reloaded. For example, you can use it to fetch information and dis-
```

```
play it on the page, or to submit a form for processing, all without changing
the full page in the browser. I'll discuss AJAX in detail in Lesson 16, "Using
JavaScript Libraries."</dd>

    <dt>When I use JavaScript, do I need to accommodate users whose browsers may
not support JavaScript or who have disabled it?</dt>
    <dd>Some estimates indicate that over 90% of Web users have JavaScript
enabled. However, unless you have a really good reason not to, you should make
accommodations for users without JavaScript. You need not offer users who don't
have JavaScript an identical experience to those who have it, but they should be
able to access your site. For example, if you run an online store, do you really
want to shut out users because of their browser configuration? </dd>
</dl>

</body>
</html>
```

Output ▶

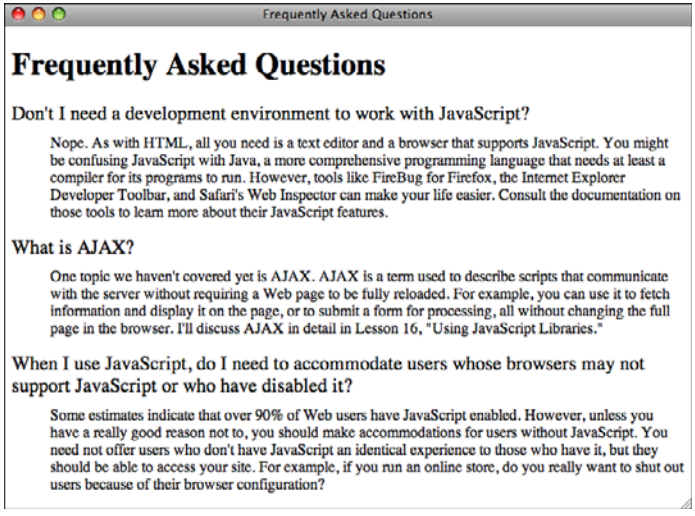


FIGURE 15.4
The FAQ page.

The page is designed so that it works perfectly well without JavaScript; all the questions and answers are displayed so that the user can read them. This is what unobtrusive JavaScript is all about.

Adding the Script After the page has been created and you've confirmed that it's working correctly, the next step is to add JavaScript into the mix. To include the script on the page, I just need to add the link to the external script to the header:

```
<script type="text/javascript" src="faq.js"></script>
```

That `<script>` tag loads and execute the script in the file `faq.js`. After the JavaScript has been added, the answers to the questions are hidden, as shown in Figure 15.5.

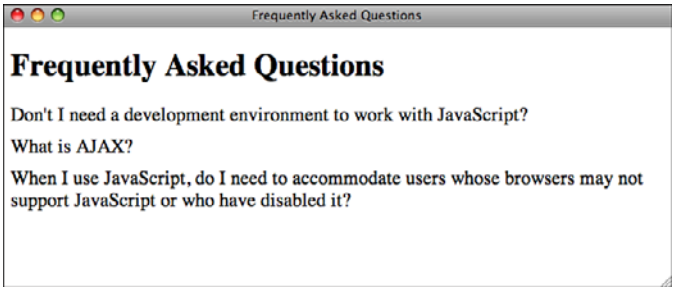


FIGURE 15.5
The FAQ page with the JavaScript included.

Here's the JavaScript contained in the `faq.js` file:

```
window.onload = function() {
    var faqList, answers, questionLinks, questions, currentNode, i, j;

    faqList = document.getElementById("faq");
    answers = faqList.getElementsByTagName("dd");

    for (i = 0; i < answers.length; i++) {
        answers[i].style.display = 'none';
    }

    questions = faqList.getElementsByTagName("dt");

    for (i = 0; i < questions.length; i++) {
        questions[i].onclick = function() {
           currentNode = this.nextSibling;
           while (currentNode) {
               if (currentNode.nodeType == "1" && currentNode.tagName == "DD") {
                   if (currentNode.style.display == 'none') {
                       currentNode.style.display = 'block';
                   }
                   else {
                       currentNode.style.display = 'none';
                   }

                   break;
               }

               currentNode = currentNode.nextSibling;
           }

           return false;
        };
    }
}
```

This JavaScript code is significantly more complex than any used previously in the book. Take a look at the first line, which is repeated here:

```
window.onload = function() {
```


This is where the unobtrusiveness comes in. Instead of calling a function using the `onload` attribute of the `<body>` tag to start up the JavaScript for the page, I assign an anonymous function to the `onload` property of the window object. The code inside the function will run when the `onload` event for the window is fired by the browser. Setting up my JavaScript this way allows me to include this JavaScript on any page without modifying the markup to bind it to an event. That's handled here.

This is the method for binding functions to events programmatically. Each element has properties for the events it supports, to bind an event handler to them, you assign the function to that property. You can do so by declaring an anonymous function in the assignment statement, as I did in this example, or you can assign the function by name, like this:

```
function doStuff() {
    // Does stuff
}
window.onload = doStuff;
```

In this case, I intentionally left the parentheses out when I used the function name. That's because I'm assigning the function itself to the `onload` property, as opposed to assigning the value returned by `doStuff()` to that property.

NOTE

When you declare an anonymous function in an assignment statement, you must make sure to include the semicolon after the closing brace. Normally when you declare functions, a semicolon is not needed, but because the function declaration is part of the assignment statement, that statement has to be terminated with a semicolon, or you'll get a syntax error when the browser tries to interpret the JavaScript.

On the next line, I declare all the variables I use in this function. JavaScript is a bit different from many other languages in that variables cannot have "block" scope. For example, in most languages, if you declare a variable inside the body of an `if` statement, that variable will go away once the if statement `is` finished. Not so in JavaScript. A variable declared anywhere inside a function will be accessible from that point onward in the function, regardless of where it was declared. For that reason, declaring all your variables at the top of the function is one way to avoid confusing bugs.

Looking Up Elements in the Document The preceding lesson discussed the `document` object a little bit, and mentioned that it provides access to the full contents of the web page. The representation of the page that is accessible via JavaScript is referred to as the *Document Object Model*, or *DOM*. The entire page is represented as a tree, starting at the root element, represented by the `<html>` tag. If you leave out the `<html>` tag, the browser will add it to the DOM when it renders the page. The DOM for this page is shown in Figure 15.6.

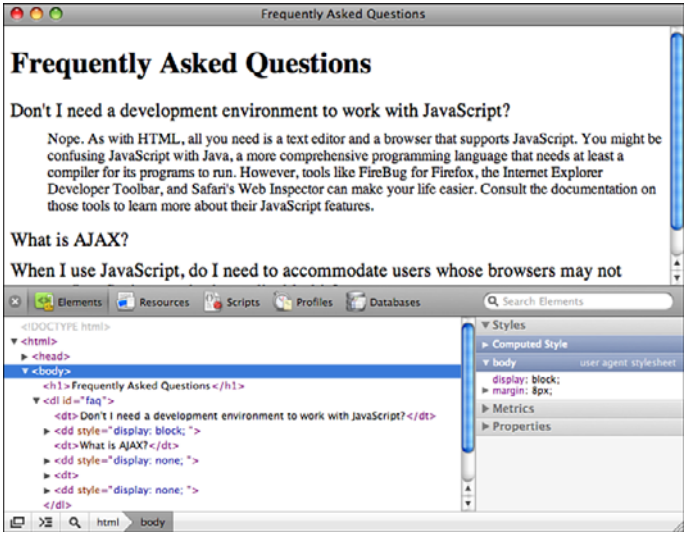


FIGURE 15.6
The DOM for the FAQ page, shown in Firebug.

There are a number of ways to dig into the DOM. The browser provides access to the parent of each element, as well as its siblings, and children, so you can reach any element that way. However, navigating your way to elements in the page that way is tedious, and there are some shortcuts available.

These shortcuts, methods that can be called on the `document` object, are listed in Table 15.1.

TABLE 15.1 Methods for Accessing the DOM

Method	Description
`getElementsByTagName(name)`	Retrieves a list of elements with the supplied tag name. This can also be called on a specific element, and it will return a list of the descendants of that element with the specified tag name.

Method	Description
`getElementById(id)`	Retrieves the element with the specified ID. IDs are assigned using the `id` attribute. This is one of the areas where JavaScript intersects with CSS.
`getElementByName(name)`	Retrieves elements with the specified value as their `name` attribute. Usually used with forms or form fields, both of which use the `name` attribute.

To set up the expanding and collapsing properly, I must hide the answers to the questions and bind an event to the questions that expands them when users click them. First, I need to look up the elements I want to modify in the DOM.

```
faqList = document.getElementById("faq");
answers = faqList.getElementsByTagName("dd");
```

The first line gets the element with the ID `faq`. That's the ID I assigned to my definition list in the markup. Then the next line returns a list of all the `dd` elements that are children of the element now assigned to `faqList`. I could skip the step of looking up the `faq` list first, but then if this page included multiple definition lists, the behavior would be applied to all of them rather than just the `faq`. This is also a useful precaution in case this JavaScript file is included on more than one page. In the end, I have a list of `dd` elements.

Changing Styles I grabbed the list of `dd` elements so that they can be hidden when the page loads. I could have hidden them using a style sheet or the style attribute of each of the `dd` elements, but that wouldn't be unobtrusive. If a user without JavaScript visited the page, the answers to the questions would be hidden, and there wouldn't be any way to reveal the answers. It's better to hide them with JavaScript.

There are two ways to hide elements with CSS, you can set the `display` property to `none` or the `visibility` property to `hidden`. Using the `display` property will hide the element completely, the `visibility` property hides the content in the element but leaves the space it takes up empty. So for this case, using the `display` property makes more sense. Every element in the document has a `style` property, and that property has its own properties for each CSS property. Here's the code that hides each of the `dd` elements:

```
for (i = 0; i < answers.length; i++) {
        answers[i].style.display = 'none';
    }
```

The `for` loop iterates over each of the elements, and inside the loop, I set the `display` property to `none`. When the page loads, the answers will be hidden.

Traversing the Document The final step is to bind the event that toggles the display of the answers to each of the questions. This is the most complex bit of code on the page. First, let me explain how the event handler works:

```
function() {
   currentNode = this.nextSibling;
   while (currentNode) {
       if (currentNode.nodeType == "1" && currentNode.tagName == "DD") {
           if (currentNode.style.display == 'none') {
               currentNode.style.display = 'block';
           }
           else {
               currentNode.style.display = 'none';
           }
           break;
       }
       currentNode = currentNode.nextSibling;
   }

   return false;
};
```

That's the function that will be used as the `onclick` handler for each of the questions. As you may remember, in the context of an event handler, `this` is the element associated with the event. The main challenge in this function is locating the answer associated with the question the user clicked on and displaying it.

To do so, the function will navigate through the DOM to find the next DD element in the DOM tree following the DT element that the user clicked on. First, I use the `nextSibling` property of `this`, and then I start a `while` loop that will iterate over each of the siblings of that element. The `while` condition ensures that the loop will run until `this` runs out of siblings.

The `nextSibling` property is a reference to the next node in the DOM tree. A node is different from an element. HTML elements are nodes, but the whitespace between tags is a node, as is the text inside a tag. So the `nextSibling` of a node might very well be the return character at the end of the line following the tag. There are a number of other properties associated with nodes as well that can be used to traverse the document. Some are listed in Table 15.2.

TABLE 15.2 Node Properties for Navigating the DOM

Method	Description
`childNodes`	An array of all the children of a node.
`firstChild`	The first child node of a node.
`innertHTML`	The markup and content inside a node. You can set this property to change the contents of a node.

Method	Description
`lastChild`	The last child of a node.
`nextSibling`	The next sibling of the node (at the same level of the DOM tree).
`parentNode`	The parent of the current node.
`previousSibling`	The node that precedes the current node at the same level of the tree.

All the properties in the table are `null` if it's not possible to traverse the DOM in that direction. For example, if a node has no child nodes, its `lastChild` property will be `null`.

Here's what happens when a user clicks one of the questions. As mentioned, a `while` loop will iterate over the siblings of the question. Inside the while loop, I check the `nodeType` and `tagName` of the current node.

The `nodeType` property contains a number that identifies what type of node is being processed. Element nodes have a node type of 1. Attributes are node type 2, and text nodes are type 3. There are 12 total node types, but those three are the main ones you'll use. In this function, I'm searching for the `<dd>` tag that follows the DT tag that contains the question. I have to check the node type before checking the `tagName` property, because only elements (which have node type 1) support the `tagName` property. If I didn't check the node type first, other node types would cause errors.

Each sibling node that follows the original `<dt>` is tested, and as soon as a `<dd>` element is found, the script toggles the visibility of that element. It then uses the `break` statement to stop executing the loop. If the node is not a `<dd>` element, then the next sibling of `currentNode` is assigned to the `currentNode` variable, and the loop is executed again. If the `<dd>` element is never found, then when there are no more siblings, the `currentNode` variable will be set to `null`, and execution of the loop will stop.

At the end, the function returns `false`:

```
questions = faqList.getElementsByTagName("dt");
for (i = 0; i < questions.length; i++) {
    questions[i].onclick = function() {
        // The actual event handling code goes here.
    }
}
```

First, I use `getElementsByTagName()` to get a list of all the `<dt>` tags that are children of `faqList`. Then I used a `for` loop to iterate over them and bind the function described previously to their `onclick` event.

Adding New Content to a Page

The last example demonstrated how to modify styles on a page. In this example, I explain how to modify the content on a page using JavaScript. You can create new elements in JavaScript and then attach them to the document in any location that you choose. You can also modify elements that are already on the page or remove elements if you need to do so.

▼ Task: Exercise 15.3: Add an Expand All/Collapse All Link to the FAQ

In this example, I add a new feature to the FAQ page presented in the previous example. In that example, I illustrated how to add new features to a page using JavaScript without modifying the markup in any way. This example will continue along those lines. I won't be making any changes at all to the markup on the page; all the changes will take place inside the JavaScript file.

In this example, I add a link to the page that expands all the questions in the FAQ, or if all the questions are already expanded, collapses all the questions. The label on the link will change depending on its behavior, and the function of the link will also change if the user individually collapses or expands all the questions.

Adding the Link to the Page Because the link functions only if the user has JavaScript enabled, I am going to add it dynamically using JavaScript. I've added a new function to the JavaScript file that takes care of adding the link, which I call from the `onload` handler for the page. The function adds more than just a link to the page. It adds a link, a `<div>` containing the link, and the `onclick` handler for the link. Here's the function, which I've named `addExpandAllLink()`:

```
function addExpandAllLink() {
  var expandAllDiv, expandAllLink, faq;

  expandAllDiv = document.createElement("div");
  expandAllDiv.setAttribute("id", "expandAll");

  expandAllLink = document.createElement("a");
  expandAllLink.setAttribute("href", "#");
  expandAllLink.setAttribute("id", "expandAllLink");
  expandAllLink.appendChild(document.createTextNode("Expand All"));

  expandAllDiv.appendChild(expandAllLink);

  expandAllLink.onclick = function() {
    var faqList, answers;
    faqList = document.getElementById("faq");
    answers = faqList.getElementsByTagName("dd");
```

```
    if (this.innerHTML == "Expand All") {
      for (i = 0; i < answers.length; i++) {
          answers[i].style.display = 'block';

      }
      this.innerHTML = "Collapse All";
    }
    else {
      for (i = 0; i < answers.length; i++) {
          answers[i].style.display = 'none';
      }
      this.innerHTML = "Expand All";
    }
    return false;
  };

  faq = document.getElementById("faq");
  faq.insertBefore(expandAllDiv, faq.firstChild);
}
```

First, I declare the variables I use in the function, and then I start creating the elements. The `createElement()` method of the `document` object is used to create an element. It accepts the element name as the argument. I create the `<div>` element and then call the `setAttribute()` method to add the `id` attribute to that element. The `setAttribute()` method takes two arguments, the attribute name and the value for that attribute. Then I create the link by creating a new `<a>` element. I set the `href` attribute to `#`, because the event handler for the link's `onclick` event will return false anyway, and I add an `id` for the link, too. To add the link text, I call the `document.createTextNode()` method:

```
expandAllLink.appendChild(document.createTextNode("Expand All"));
```

I pass the results of that method call to the `appendChild()` method of `expandAllLink`, which results in the text node being placed inside the `<a>` tag. Then on the next line I append the link to the `<div>`, again using `appendChild()`. The last thing to do before appending the `<div>` to an element that's already on the page (causing it to appear) is to add the `onclick` handler to the link.

I'm again attaching the `onclick` handler using an anonymous function, as I did in the previous example. In this case, I use the same technique I used in the previous example, obtaining a reference to the `<div>` with the ID `faq` and then retrieving a list of `<dd>` elements inside it.

At that point, I inspect the contents of `this.innerHTML`. In an event handler, `this` is a reference to the element upon which the event was called, so in this case, it's the link. The `innerHTML` property contains whatever is inside that element, in this case, the link text. If the link text is "Expand All," I iterate over each of the answers and set their

display property to block. Then I modify the this.innerHTML to read "Collapse All". That changes the link text to Collapse All, which not only alters the display, but also causes the same function to hide all the answers when the user clicks on the link again. Then the function returns false so that the link itself is not processed.

When the onclick handler is set up, I add the link to the document. I want to insert the link immediately before the list of frequently asked questions. To do so, I get a reference to its using getElementById() and then use insertBefore() to put it in the right place:

```
faq = document.getElementById("faq");
faq.insertBefore(expandAllDiv, faq.firstChild);
```

Table 15.3 contains a list of methods that can be used to modify the document. All of them are methods of elements.

TABLE 15.3 Methods for Accessing the DOM

Method	Description
`appendChild(element)`	Adds the element to the page as a child of the method's target
`insertBefore(new, ref)`	Inserts the element `new` before the element `ref` on the list of children of the method's target.
`removeAttribute(name)`	Removes the attribute with the supplied name from the method's target
`removeChild(element)`	Removes the child of the method's target passed in as an argument
`replaceChild(inserted, replaced)`	Replaces the child element of the method's target passed as the `inserted` argument with the element passed as the parameter `replaced`
`setAttribute(name, value)`	Sets an attribute on the method target with the name and value passed in as arguments

There's one other big change I made to the scripts for the page. I added a call to a new function in the handler for the click event for the questions on the page:

```
updateExpandAllLink();
```

That's a call to a new function I wrote, which switches the Expand All / Collapse All link if the user manually collapses or expands all the questions. When the page is opened, all the questions are collapsed, and the link expands them all. After the user has

expanded them all one at a time, this function will switch the link to `Collapse All`. The function is called every time the user clicks on a question. It inspects the answers to determine whether they are all collapsed or all expanded, and adjusts the link text accordingly. Here's the source for that function:

```
function updateExpandAllLink() {
  var faqList, answers, expandAllLink, switchLink;

  faqList = document.getElementById("faq");
  answers = faqList.getElementsByTagName("dd");
  expandAllLink = document.getElementById("expandAllLink");
  switchLink = true;

  if (expandAllLink.innerHTML == "Expand All") {
    for (i = 0; i < answers.length; i++) {
      if (answers[i].style.display == 'none') {
        switchLink = false;
      }
    }

    if (switchLink) {
      expandAllLink.innerHTML = "Collapse All";
    }
  }
  else {
    for (i = 0; i < answers.length; i++) {
      if (answers[i].style.display == 'block') {
        switchLink = false;
      }
    }

    if (switchLink) {
      expandAllLink.innerHTML = "Expand All";
    }
  }
}
```

This function starts with some setup. I declare the variables I will be using, and retrieve the elements I need to access from the DOM. I also set the variable `switchLink` to true. This variable is used to track whether I need to switch the link text in the `Expand All` link. When everything is set up, I use an `if` statement to test the state of the link. If the link text is set to `Expand All`, it checks each of the answers. If any of them are hidden, it leaves the link as is. If all of them are displayed, it changes the link text to `Collapse All`. If the link text is already `Collapse All`, the test is the opposite. It switches the link text to `Expand All` if all the questions are hidden.

Summary

This lesson demonstrated a number of common tasks associated with programming in JavaScript. It illustrated how to access the values in forms and check them for errors. It also explained how you can manipulate the styles on a page and even the contents of a page using JavaScript. The final two examples were written in a style referred to as unobtrusive JavaScript, which involves writing JavaScript in such a way that the page still works even if the user has disabled JavaScript, or their browser does not offer JavaScript support. JavaScript is used to enhance the user's experience, but the functionality of the page is not dependent on JavaScript. This approach is generally favored as the preferable way to write JavaScript these days. It separates JavaScript code from the markup on the page and ensures support for the largest number of users, including users with mobile browsers that may not support JavaScript functionality.

In the next lesson, I demonstrate how to perform some of the same kinds of tasks as I did in this lesson using the popular JavaScript library jQuery, and explain the role jQuery and other libraries like it can play in making it easier to add JavaScript functionality to web pages.

Workshop

The following workshop includes questions, a quiz, and exercises related to the uses of JavaScript.

Q&A

Q **Can you point me in the direction of more scripts that I can integrate with my pages?**

A Sure, there are lots of sites with prepackaged JavaScript programs that you can use on your pages. You might try The JavaScript Source at http://javascript.internet.com/, or JavaScript.com at http://www.javascript.com/.

Q **Is there a way to incorporate form validation into my page so that errors are displayed in the form rather than in a pop-up?**

A Yes, using the techniques shown in the other examples in this lesson, you could modify the document itself when validating the form. The trick is to modify the DOM after validating the values, as opposed to displaying the message using the `alert()` method. In the next lesson, I provide an example of how to display errors in forms themselves using a JavaScript library.

Quiz

1. What happens whenever a user clicks a link, button, or form element on a web page?
2. In an event handler, what does `this` refer to?
3. What kinds of nodes on a page can be associated with properties like `nextChild` and `previousChild`?
4. How does form validation with JavaScript conserve server resources?

Quiz Answers

1. Whenever a user clicks a link, a button, or any form element, the browser generates an event signal that can be captured by one of the event handlers mentioned in the previous lesson.
2. In event handlers, `this` is a reference to the element on which the event was called. So in an event handler for the `onclick` event of a link, `this` would refer to the link that the user clicked on.
3. Nodes in the DOM can include HTML elements, text inside HTML elements, and even whitespace between elements.
4. JavaScript enables you to do error checking in forms on the browser side before the form is ever submitted to the server. A script must access the server before it can determine the validity of the entries on a form. (Note that even if you use JavaScript form validation you must validate user input on the server, too, because users can bypass the JavaScript if they choose.)

Exercises

1. Change the HTML validation example to add error messages to the page above the form when validation fails.
2. Add a Preview button to the form validation example that displays the values the user entered below the form.
3. Modify the FAQ example so that users can click a link for each question to remove that question from the page entirely.

This page intentionally left blank

LESSON 16

Using JavaScript Libraries

In the previous two lessons, I explained the ins and outs of the JavaScript language and the browser as a platform for writing programs. In this lesson, you're going to learn that in the previous two lessons, all the example programs were written the hard way. JavaScript libraries make taking advantage of JavaScript much easier, and in this lesson, you'll learn about the advantages of using them.

In this lesson, you'll learn about the following:

- What JavaScript libraries are and why you might want to use one
- Which libraries people are currently using
- How to use the jQuery library in your pages
- How to bind events using jQuery
- How to manipulate styles on a page
- How to change the content of a page
- How to fetch content from an external source using AJAX

What Are JavaScript Libraries?

In this book, I've talked about browsers and incompatibilities between them. The popular browsers differ in their support for HTML and CSS, and also in their support for JavaScript. Unlike CSS and HTML, though, JavaScript can actually be used to solve the problem of implementations of JavaScript that differ in small but important ways. You can write programs that detect which browser is being used, or even the specific capabilities of the browser being used and then add logic to make sure the program works correctly for whatever environment that it's in.

For example, some browsers allow you to retrieve elements from the document by class name using the `getElementsByClassName()` method, and others do not. If your script depends on that method, it will break in some browsers. You can work around the problem by checking to see whether the method exists before you use it, and if it doesn't, using another technique that works in the browsers that don't support it.

JavaScript libraries were created by people who had to do this sort of thing too many times and decided to package up all these kinds of workarounds to create a simpler interface to common functionality that hides all the incompatibilities of the various browsers. In doing so, the authors also added many other features to make life more convenient for JavaScript developers. The most popular JavaScript libraries all make it easier to bind code to events, select elements on the page to act on in your programs, and even make calls to the server from JavaScript to dynamically change elements on the page, using a technique referred to as AJAX.

You might have noticed that I am referring to JavaScript libraries as a group. That's because there are a number of libraries that provide roughly the same set of features. They were all independently developed and work differently from one another; each has its own set of advantages and disadvantages. If you're starting from scratch, choosing between them is a matter of taste.

Reviewing the Popular JavaScript Libraries

In this lesson, the examples are written in jQuery, but they could have been written in any of the popular JavaScript libraries. Here's a quick overview of the most popular libraries that people are using. All these libraries are open source, free software that you can use on your own website, even if it's a commercial site.

jQuery

jQuery is the most popular JavaScript library right now. It is widely used because it's easy to learn and because it makes it simple for even a new user to accomplish a lot with relatively little code, as you'll see in this lesson. The advance that jQuery introduced is the ability to use CSS selectors in JavaScript. One of the toughest problems for JavaScript programmers had been coming up with easy ways to target specific elements on the page when manipulating page content—jQuery enabled programmers to use a technique they were already familiar with. Other JavaScript libraries have since added support for CSS selectors, as well. Whenever any of the JavaScript libraries introduce a powerful new feature, it makes its way to the other libraries relatively quickly.

You can download jQuery and browse the documentation at http://jquery.com. jQuery is also popular because of the wide variety of tools for developers that have been built on top of it. For example, a variety of advanced user interface components have been built using jQuery and are available at http://jqueryui.com/. These components include things such as date pickers, sortable lists, and dialog boxes that are widely used and can be painful to create from scratch.

Dojo

Dojo is a JavaScript library that's popular with programmers. It has been widely adopted by corporate websites and as a component in packaged Web applications. Dojo has a bit more built in functionality than jQuery; for example, it includes data grid functionality that enables you to create tables that are sortable on-the-fly. Like jQuery, Dojo has a separate UI library, called Dijit. You can download Dojo and read more about it at http://www.dojotoolkit.org/.

Yahoo! UI

Yahoo! UI, or YUI for short, is a JavaScript library created by Yahoo! to use on its website. They've shared it with the rest of the world. You can find out more about it at http://developer.yahoo.com/yui/. Yahoo! UI provides roughly the same set of features as the other popular JavaScript library and has the added benefit of robust, professionally written documentation. One disadvantage of YUI is that it lacks the community of third-party developers that libraries like jQuery and Dojo have, so there are fewer plug-ins and extensions available.

Prototype

Prototype is, in some ways, the original JavaScript library. Prototype slipped a bit in popularity when jQuery arrived due to jQuery's ease of use, but has since added most of

people's favorite jQuery features. It is perhaps most famous for a library of effects built using prototype, called Scriptaculous. Scriptaculous is a collection of animations you can apply to animate elements on web pages. No longer did items you removed from the page just blink out of existence; they could shrink, or fade away, or explode. Similar libraries of effects have since been written for other libraries, but Scriptaculous showed the world what could be accomplished in terms of visual effects using only JavaScript and CSS. You can get Prototype at http://www.prototypejs.org/, and you can check out Scriptaculous at http://script.aculo.us/.

Other Libraries

There are a number of other JavaScript libraries, too. Here's a list of a few others you may encounter, or want to check out:

- **Google Web Toolkit**—Enables you to create JavaScript applications using Java and compile them into JavaScript (http://code.google.com/webtoolkit/)
- **Midori**—Optimized for quick download (http://www.midorijs.com/)
- **MochiKit**—Focused on stability and documentation (http://www.mochikit.com/)
- **MooTools**—One of the most mature frameworks, optimized for size (http://mootools.net/)

Getting Started with jQuery

Entire books are published about each of the popular JavaScript libraries, so it would be foolish to try to cover them all in this lesson. Instead, I'm going to focus on introducing jQuery. I've chosen it mainly because it's the easiest library to get started with, especially if you already know CSS. Even if you don't wind up using jQuery, you'll still get an idea of how JavaScript libraries work by reading this section. You'll just have to follow up by digging into the documentation to learn how to apply the same concepts with the library that you use instead.

jQuery is a regular JavaScript file that you can include in your page using the `<script>` tag. To get started, download your own copy at http://jquery.com. After you've downloaded jQuery, you can start using it in your pages. The easiest way to included in a page, especially for local testing, is to rename the downloaded file to `jquery.js` and put it in the same directory as the HTML page:

```
<script type="text/javascript" src="jquery.js"></script>
```

NOTE

The file you download will have a different name than `jquery.js`, because it will include the jQuery version number. You'll have to rename it as `jquery.js` or use the full filename in your `<script>` tag. You can download jQuery in production or development configurations. The production configuration is "minified"—compressed so that it downloads as quickly as possible. Unfortunately, the minified file is unreadable by humans, so if you think you may need to look at the jQuery source, download the development version. Just be sure to replace it with the minified version when you make your site live.

Your First jQuery Script

jQuery is built around the idea of selecting objects on the page and then performing actions on them. In some ways, it's similar to CSS. You use a selector to define an element or set of elements, and then you write some code that is applied to those elements. Here's an example of a page that includes a simple jQuery script:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery Example</title>
  <script src="jquery.js"></script>
  <script>
    $(document).ready(function() {
      $("a").click(function(event) {
        alert("You clicked on a link to " + this.href );
      });
    });
  </script>
</head>
<body>
  <a href="1.html">A link</a>
</body>
</html>
```

The first `<script>` tag includes the external jQuery script. The second contains the script I wrote. This script causes an alert to be displayed whenever a user clicks a link. As you can see, the JavaScript is unobtrusive—all of it is inside the `<script>` tag. I'll break it down line by line.

The first line is important, because you'll see it or a variation of it in nearly every jQuery script:

```
$(document).ready(function() {
```

First, `$` is the name of a function provided by jQuery, and `document` is an argument to that function. The `$` function selects the set of elements matched by the selector provided as an argument. In this case, I've passed `document` as the argument, and it matches the `document` object—the root object of the page's document object model. Usually, the selector is a CSS selector, but the `document` object is an alternative that you can use, as well.

To the right, you see a call to the `ready` method, which is applied to the elements that the selector matches. In this case, it's the `document` object. jQuery provides convenience methods for binding events to objects or elements, and in this case, will be used to bind an anonymous function to the document's `ready` event. It's roughly equivalent to `window.onload`, the method call I used in the previous lesson to cause JavaScript to execute when the page loads, but it works a bit differently. The `window.onload` event doesn't "fire" (call any methods that are bound to it) until the page has fully loaded. This can be a problem for pages that contain large images, for example. The JavaScript code won't run until the images load, and that could lead to strange behavior for your users.

jQuery's `document.ready` event, on the other hand, fires when the DOM for the page has been fully constructed. This can cause the JavaScript to run a bit earlier in the process, while images are being downloaded. With jQuery it's customary to perform all the work you want to occur when the page loads within an anonymous function bound to the `document.ready` event. It's so common that a shortcut is provided to make doing so even easier. The previous line can be rewritten as follows:

```
$(function() {
```

jQuery knows that you intend to bind the function to the `document.ready` event. Here's the code that's bound to the event:

```
$("a").click(function(event) {
  alert("You clicked on a link to " + this.href );
});
```

This code binds a function that prints an alert message containing the URL of the link that the user clicked on to every link on the page. In this case, I use `"a"` as the selector I pass to jQuery, and it works exactly as it does with CSS. The `click()` method binds a function to the `onclick` event for all the elements matched by the selector it's called on, in this case, all the `<a>` tags on the page.

Binding Events

In the introductory script, I provided one example of event binding. The biggest advantage of binding events using jQuery (or other popular JavaScript libraries) is that they are nondestructive. There's a big problem with writing code like this:

```
window.onload = function () {  // Do stuff. }
```

If some other script also binds an event to the `window.onload` event, one of the two will win, and the other event binding will be ignored. This can be a big problem on web pages that include multiple external JavaScript files. jQuery (and other libraries) bind the events in such a way that multiple handlers can be bound to a single event without unbinding the other handlers. The functionality may still conflict, but the event binding itself will work.

Another advantage of jQuery is that you can bind handlers to multiple elements at once. For example, the following code would disable all the links on the page:

```
$("a").click(function(event) { event.preventDefault(); }
```

In this case, I added the `event` parameter to my event handler so that I can refer to that event within the handler. I then call the `event.preventDefault()` method, which is part jQuery, to tell the browser not to do whatever the default action of the event was. In this case, the browser won't follow the link because that action was prevented.

Here's a more useful example. Let's say that I want links to other sites to open in a new window, and all the links to external sites use a fully qualified URL, whereas local links do not. I could use the following event handler:

```
$(function () {
  $("a").click(function (event) {
    if (null != this.href && this.href.length > 4
      && this.href.substring(0, 4) == "http") {
      event.preventDefault();
      window.open(this.href);
    }
  });
});
```

In this case, I examine the `href` attribute of the link the user clicked. If it starts with "http," I prevent the default action and open a new window with the link. If it doesn't, the default action is allowed to continue. Instead of adding special classes to external links on the page or using the `onclick` attribute for each of them to open new windows, I just used jQuery's selector functionality and a bit of programming to take care of it for me.

jQuery provides methods for binding most events to jQuery objects. For a more full list of events jQuery supports, see http://api.jquery.com/category/events/.

Modifying Styles on the Page

Another powerful feature of jQuery is that it enables you to modify styles on the page on-the-fly. jQuery enables you to modify the page styles indirectly through convenience methods that hide and show elements, for example, and also enables you to directly modify the page styles.

Hiding and Showing Elements

For example, you can hide and show elements easily based on activity by the user. Here's an example page that swaps out two elements whenever they are clicked:

```
<!DOCTYPE html>

<html>
<head>
  <title>Anchors</title>
  <script src="jquery-1.4.2.min.js" type="text/javascript" charset="utf-
8"></script>
  <script type="text/javascript" charset="utf-8">
  $(function () {
    $("#closed").hide();
    $("#open, #closed").click(function (event) {
      $("#open, #closed").toggle();
    });
  });
  </script>
</head>
<body>

<div id="open" style="padding: 1em; border: 3px solid black; font-size:
300%;">We are open</div>

<div id="closed" style="padding: 1em; border: 3px solid black; font-size:
300%;">We are closed</div>

</body>
</html>
```

The page contains two `<div>`s, one containing the text "We are closed" and one containing the text "We are open." In the event handler for the document's ready state, I hide the `<div>` with the ID `closed`:

```
$("#closed").hide();
```

That method sets the `display` style of the elements matched by the selector to `none` so that when the page finishes loading, that `<div>` will not be visible, as shown in Figure 16.1.

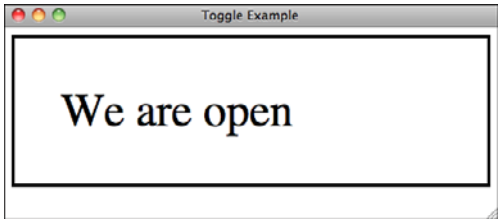


FIGURE 16.1
The initial state of the page. "We are closed" is hidden.

Then I bind an event handler to the `onclick` event of those `<div>`s containing the following code:

```
$("#open, #closed").toggle();
```

As you can see, this selector matches both the IDs `open` and `closed`, and calls the `toggle()` method on each of them. That method, provided by jQuery, displays hidden items and hides items that are being displayed. So, clicking the `<div>` will cause the other `<div>` to appear and hide the one you clicked. After you click the `<div>` and the two elements have been toggled, the page appears as shown in Figure 16.2.

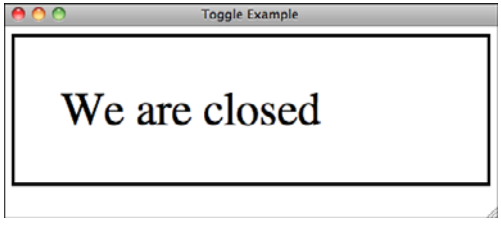


FIGURE 16.2
The initial state of the page. "We are closed" is hidden.

Retrieving and Changing Style Sheet Properties

You can also modify styles on the page directly. If I change the event handler in the previous example to contain the following code, the text will be underlined when the user clicks the `<div>`, as shown in Figure 16.3:

```
$(this).css("text-decoration", "underline");
```

FIGURE 16.3
The text is underlined after the user clicks on the `<div>`.

jQuery enables you to manipulate any styles on the page in this fashion. You can also retrieve the values of CSS properties using the `css()` method, just don't leave out the argument. If I instead change the body of the event handler to the following, the browser will display the current font size used in the `<div>` that the user clicked:

```
alert("The font size is " + $(this).css("font-size"));
```

A browser window with the alert displayed appears in Figure 16.4.

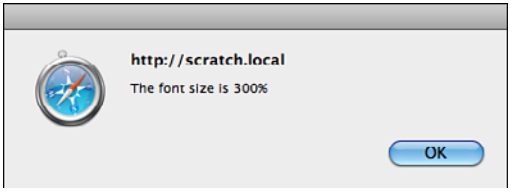


FIGURE 16.4
An alert box displaying the value of a CSS property.

Using these techniques, you can build pages with expanding and collapsing lists, add borders to links when users mouse over them, or allow users to change the color scheme of the page on-the-fly.

Modifying Content on the Page

Not only can you modify the styles on the page using jQuery, but you can also modify the content of the page itself. It provides methods that enable you to remove content from the page, add new content, and modify existing element, too.

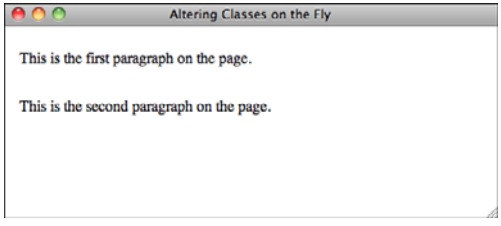
Manipulating Classes

jQuery provides a number of methods for manipulating the classes associated with elements. If your page already has a style sheet, you might want to add or remove classes from elements on-the-fly to change their appearance. In the following example page, shown in Figure 16.5, the class highlighted is added to paragraphs when the mouse is moved over them and removed when the mouse moves out:

```
<!DOCTYPE html>
<html>
<head>
       <title>Altering Classes on the Fly</title>
       <script src="jquery-1.4.2.min.js" type="text/javascript"></script>
       <script type="text/javascript">
       $(function () {
         $("p").mouseenter(function () {
      $(this).addClass("highlighted");
    });
```

```
    $("p").mouseleave(function () {
      $(this).removeClass("highlighted");
    });
  })
      </script>
      <style type="text/css" media="screen">
       p { padding: .5em;}
       p.highlighted { background: yellow; }
      </style>
</head>
<body>

<p>This is the first paragraph on the page.</p>
<p>This is the second paragraph on the page.</p>

</body>
</html>
```


FIGURE 16.5
No paragraphs are highlighted initially.



On this page, I have two paragraphs that have no classes assigned to them by default. I also have a style sheet that applies a yellow background to any paragraph with the class `highlighted`. Most important, I have the following two event handlers:

```
$("p").mouseenter(function () {
  $(this).addClass("highlighted");
});
$("p").mouseleave(function () {
  $(this).removeClass("highlighted");
});
```

In this example, I use the jQuery `mouseenter` and `mouseleave` events to fire events whenever the user moves their mouse over or away from a paragraph. As you can see in Figure 16.6, when the user's mouse is over the paragraph, the class `highlighted` is applied to it. When the mouse moves away, the class is removed.

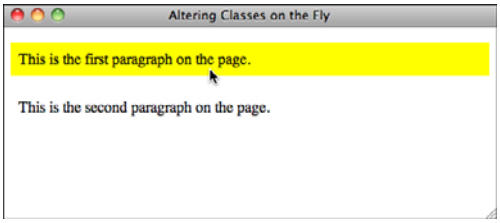


FIGURE 16.6
Paragraphs are highlighted when the mouse is over them.

You can use jQuery's `toggleClass()` method to reverse the state of a particular class on an element. In the following example, the elements in the list are highlighted the first time the user clicks them, and the highlighting is removed the next time the user clicks them. All that's required is to toggle the `highlighted` class with each click:

```
<!DOCTYPE html>
<html>
<head>
  <title>Altering Classes on the Fly</title>
  <script src="jquery-1.4.2.min.js" type="text/javascript" charset="utf-
8"></script>
  <script type="text/javascript" charset="utf-8">
    $(function () {
      $("li").click(function () {
        $(this).toggleClass("highlighted");
      });
    });
  </script>
  <style type="text/css" media="screen">
    li.highlighted { background: yellow; }
  </style>
</head>
<body>

<ul>
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
  <li>Four</li>
</ul>

</body>
</html>
```

Finally, jQuery can check for the presence of a class using the `hasClass()` method. If I change the body of the event handler in the previous example to the following function, the first time the user clicks a list item, the `highlighted` class will be applied. The second time, an alert (shown in Figure 16.7) will be displayed indicating that the item is already highlighted:

```
$("li").click(function () {
  if (!$(this).hasClass("highlighted")) {
    $(this).addClass("highlighted");
  }
  else {
    alert("This list item is already highlighted.");
  }
});
```

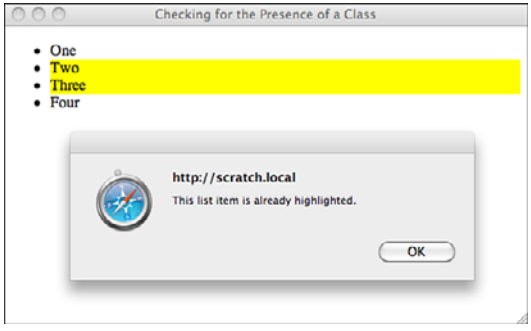


FIGURE 16.7
An alert is displayed when users click a paragraph the second time.

In this example, I use the `hasClass()` method to determine whether the class is already present. If it isn't, I add it. If it is, I display the alert.

Manipulating Form Values

You can also use jQuery to modify the contents of form fields. The `val()` method can be used to both retrieve the value of form fields and to modify them. In many cases, websites put an example of the input that should be entered into a form field in the field until the user enters data. In the following example, the form starts with example data in the field, but it's removed automatically when the user focuses on the field. If the user doesn't enter any data, the example data is restored. Figure 16.8 shows the initial state of the page.

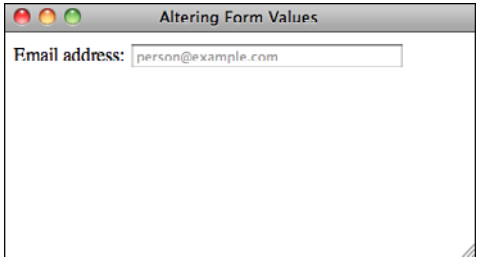


FIGURE 16.8
When the page loads, the sample content appears in the form field.

```
<!DOCTYPE html>
<html>
<head>
  <title>Altering Form Values</title>
  <script src="jquery-1.4.2.min.js" type="text/javascript" charset="utf-
8"></script>
  <script type="text/javascript" charset="utf-8">
  $(function () {
    $("input[name='email']").focus(function () {
      if ($(this).val() == "person@example.com") {
        $(this).val("");
        $(this).css("color", "black");
      }
    });

    $("input[name='email']").blur(function () {
      if ($(this).val() == "") {
        $(this).val("person@example.com");
        $(this).css("color", "#999");
      }
    });
  });
  </script>
  <style type="text/css" media="screen">
    input[name="email"] { color: #999; }
  </style>
</head>
<body>

  <form>
    <label>Email address: <input name="email" value="person@example.com"
size="40" /></label>
  </form>

</body>
</html>
```

Again, I use two event handlers in this example. The event handlers are new, as is the selector. Here's one of them:

```
$("input[name='email']").focus(function () {
```

In this case, I'm using a selector that's based on an attribute value. It matches an input field where the `name` attribute is set to `email`, and it binds to the `focus` event. This event fires when the user places the cursor in that field. The event handler for the focus event does two things: sets the value of the field to an empty string and changes the color from gray to black, but only if the value is `person@example.com`. If it's something else, it's a value the user entered and should be left alone. Figure 16.9 shows what the form looks like when the user initially clicks in the field.

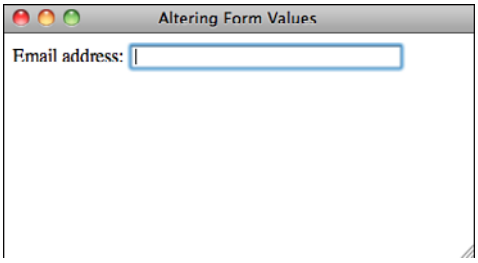


FIGURE 16.9
The contents of the email field are removed when the user clicks in it.

The other event handler is bound to the blur event, which fires when the cursor leaves the field. If the field has no value, it changes the color back to gray and puts the example input back into the field.

Manipulating Attributes Directly

You can also use jQuery to manipulate the attributes of elements directly. For example, disabling a form field entirely requires you to modify the `disabled` attribute of that field.

I've added a Submit button to the form from the previous example, and set it to `disabled`. Here's the new form:

```
<form>
  <label>Email address: <input name="email" value="person@example.com"
size="40">
  <input id="emailFormSubmit" type="submit" disabled>
</form>
```

Figure 16.10 shows the form with the sample content and the disabled Submit button.

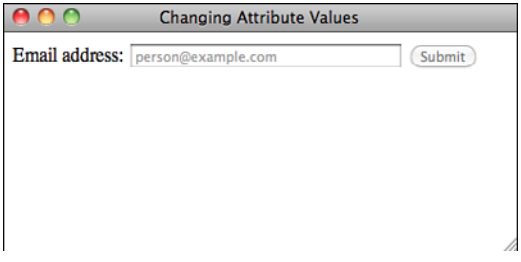


FIGURE 16.10
This form contains sample content, and the Submit button is disabled.

I only want to let users click the Submit button if they've already entered an email address. To add that check, I need to add a bit of code to the `blur` event for the email field, as shown:

```
$("input[name='email']").blur(function () {
  if ($(this).val() == "") {
    $(this).val("person@example.com");
```

```
        $(this).css("color", "#999");
        $("#emailFormSubmit").attr("disabled", "disabled");
      }
      else {
        $("#emailFormSubmit").removeAttr("disabled");
      }
    });
```

If the user leaves the field having set a value, the `disabled` attribute is removed from the Submit button, as shown in Figure 16.11. If the user leaves the field without having entered anything, the `disabled` attribute is added, just in case it was previously removed.

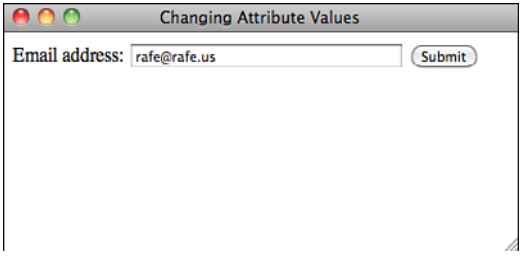


FIGURE 16.11
The Submit button is no longer disabled after an email address is entered.

Adding and Removing Content

jQuery provides a number of methods that can be used to manipulate the content on the page directly. Here's a more complex example that demonstrates several ways of manipulating the content on a page—users can add new content to the page, remove content from the page, and even wipe out all the content inside an element in one click. The initial page appears in Figure 16.12.

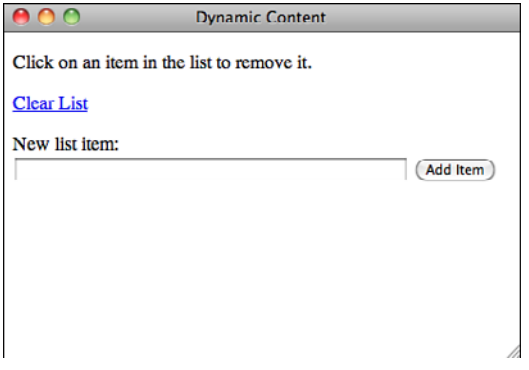


FIGURE 16.12
A page that allows you to add and remove content on-the-fly.

I'll start with the markup for the page. First, I need a list. In this example, the user will be able to add elements to the list and remove elements from the list. All I need is an empty list with an ID:

```
<ul id="editable">
</ul>
```

Next, I have a form that enables users to add a new item to the end of the list. It has a text input field and a Submit button:

```
<form id="addElement">
  <label>New list item: <input name="liContent" size="60" /></label>
  <input type="submit" value="Add Item" />
</form>
```


And finally, I've added a link that removes all the elements the user has added to the list:

```
<p><a id="clearList" href="#">Clear List</a></p>
```

The action is on the JavaScript side. Let's look at each of the event handlers for the page one at a time. First, the event handler for the Clear List link

```
$("#clearList").click(function (event) {
    event.preventDefault();
    $("#editable").empty();
});
```

This event handler prevents the default action of the link (which would normally return the user to the top of the page) and calls the `empty()` method on the list, identified by selecting its ID. The `empty()` method removes the contents of an element.

Next is the event handler for the form, which enables users to add new items to the list:

```
$("#addElement").submit(function (event) {
  event.preventDefault();
  $("#editable").append("<li>"
    + $("#addElement input[name='liContent']").val() + "</li>");
  $("#addElement input[name='liContent']").val("");
});
```

I bind this handler to the `submit` event for the form, just as I did in the previous example. First, it prevents the form submission from completing its default action—submitting the form to the server. Then I append the content to the list. I selected the list using its ID, and then I call the `append()` method, which adds the content in the argument just inside

the closing tag for the elements that match the selector. In this case, I put the value in the text field, which I obtain using the `val()` method that you've already seen, inside an opening and closing `<li>` tag, and pass that to the `append()` method. I also remove the content from the text field because it has been appended to the list. Figure 16.13 shows the list once a few elements have been added.

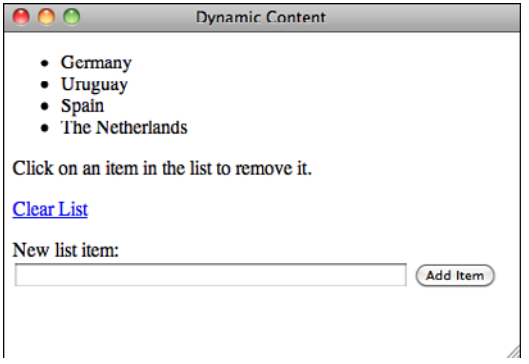


FIGURE 16.13
A page that allows you to add and remove content on-the-fly.

Finally, I allow users to remove items from the list by clicking them. There's one trick here. As you've seen, to do so I'll need to use the `click` handler for the `<li>` elements in the list. In this case, there's a catch. When the page loads and the `document.ready` event for the page initially fires, there are no elements in the list to bind events to. Fortunately, jQuery provides a way to set up an event handler so that it's automatically bound to newly created elements on the page. Here's the code:

```
$("#editable li").live('click', function () {
  $(this).remove();
});
```

As you can see, the event binding is slightly different here. Instead of using the `click()` method, I've used the `live()` method. This indicates that I want to monitor changes to the page and perform the event binding that follows any time an element matching the selector is added. The first argument is the name of the event to bind—it's the name of the event to be bound, placed in quotation marks. The second is the event handler as it would normally be written. The `live()` method is one of the most powerful features of jQuery because it enables you to automatically treat dynamically generated content the same way you'd treat content that's on the page at the time that it loads.

Here's the full source for the page:

```
<!DOCTYPE html>

<html>
```

```
<head>
        <title>Altering Classes on the Fly</title>
        <script src="jquery-1.4.2.min.js" type="text/javascript" charset="utf-
8"></script>
        <script type="text/javascript" charset="utf-8">
        $(function () {
          $("#editable li").live('click', function () {
            $(this).remove();
    });

    $("#clearList").click(function (event) {
      event.preventDefault();
      $("#editable").empty();
    });

    $("#addElement").submit(function (event) {
      event.preventDefault();
      $("#editable").append("<li>" + $("#addElement
input[name='liContent']").val() + "</li>");
       $("#addElement input[name='liContent']").val("");
    });
  });
        </script>
</head>
<body>

<ul id="editable">
</ul>

<p>Click on an item in the list to remove it.</p>

<p><a id="clearList" href="#">Clear List</a></p>

<form id="addElement">
  <label>New list item: <input name="liContent" size="60" /></label>
  <input type="submit" value="Add Item" />
</form>

</body>
</html>
```

There are other methods for adding content in different locations in relation to a selected element. For example, if I change the `append()` call to `prepend()`, new items will be added to the top of the list rather than the bottom. You can also use the `before()` method to add content before another element and the `after()` element to add it after. The difference is that when you use those methods, the content is placed outside the tags matched by the selector, rather than inside those tags.

Special Effects

It can be a little jarring when elements just appear or disappear instantly. Most JavaScript libraries, including jQuery, provide a library of effects that enable you to animate transitions on the page when items appear, disappear, or move. jQuery has a few basic effects built in to the core library. Supplemental effects are also available as part of jQuery UI, which you can obtain at http://jqueryui.com/.

The four effects that are part of jQuery are fade in, fade out, slide up, and slide down. I'm going to build on the previous example to show you how they can be used to soften the transitions when you add items to the page or remove items from it. Adding the effects to the page just requires a few small tweaks to the event handlers that I already created.

The first effect I added applies the fade-out effect when users click a list item to remove it. To cause an element to fade out, you call the `fadeOut()` method on the results of a selector that matches that element. Here's the code:

```
$("#editable li").live('click', function () {
  $(this).fadeOut('slow', function() { $(this).remove() });
});
```

When you call `fadeOut()`, it sets the `display` property for the element to `none`—essentially, it's a fancy replacement for `hide()`. Figure 16.14 shows a list item that's in the process of fading out.

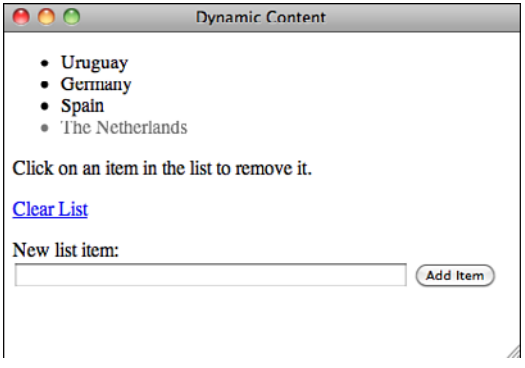


FIGURE 16.14 The jQuery fade-out effect in progress.

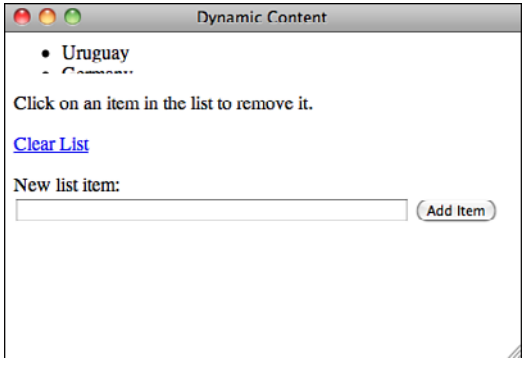
In this case, I want to actually remove the element from the page entirely. To do so, I need a callback, which is included as the second argument to `fadeOut()`. The callback is run whenever the animation is complete, and in this case, removes the element from the page. The first argument is used to specify the speed of the animation. Setting it to `slow`

means that it will take 600 milliseconds to complete. By default, the animation takes 400 milliseconds. You can also set it to `fast` (200 milliseconds), or you can enter a number of milliseconds yourself.

I've also updated the event handler for the Clear List link. In this case, I use the slide-up effect, shown in Figure 16.15, when the list is cleared. Here's the updated event handler:

```
$("#clearList").click(function (event) {
    event.preventDefault();
    $("#editable").slideUp('slow', function () {
      $("#editable").empty()
      $("#editable").show();
    });
});
```


FIGURE 16.15
The jQuery slide-up effect.



The changes here are similar to those for the previous event handler. After the animation is complete and the list is hidden, I call the `empty()` method to remove the contents of the list and then call `show()` on the now hidden list so that when the user adds new elements to it, the list will be visible.

Finally, I want the new items I add to the list to fade in rather than just appearing. Here's the updated event handler with the `fadeIn()` call included:

```
$("#addElement").submit(function (event) {
      event.preventDefault();
      var content = "<li>" + $("#addElement input[name='liContent']").val() +
"</li>";
      $(content).hide().appendTo("#editable").fadeIn('slow').css("display",
"list-item");
    });
```

This event handler is a little bit more complex. First, I initialize a new variable with the content to add to the page, just to make the code a little more readable. Then, I go

through all the steps required to fade the new content in. At this point, I should explain one of the other nifty features of jQuery—method chaining. Nearly all jQuery methods return the object of the method. So, if I use `hide()` to hide something, the method returns whatever it was that I hid. This makes it convenient to call multiple methods on the same object in succession.

In this case, I call `hide()`, `appendTo()`, `fadeIn()`, and `css()` on the jQuery object representing the new content that I created. First, I pass the content variable to `$()`; this allows me to call jQuery's methods on the content. Then, I call `hide()` on it so that it doesn't appear instantly when I append it to the list.

After that, I use `appendTo()` to append it to the list. The difference between `append()` and `appendTo()` is that with `append()`, the object of the method is the selector that represents the container, and the method parameter is the content to be appended, whereas with `appendTo()`, the content to be appended is the object and the selector for the container is the method parameter. In this case, using `appendTo()` makes it easier to chain all of these method calls.

After I've appended the hidden content to the list, I call `fadeIn('slow')` to make it gradually appear. Then, finally, I call `css("display", "list-item")` on the new content, because when `fadeIn()` is done, it sets the `display` property for the list item to `block`, which causes the bullet for the list item not to appear in some browsers. Setting the `display` property to `list-item` ensures that a bullet is displayed.

AJAX and jQuery

One of the primary reasons programmers started adopting JavaScript libraries was that they made it much easier to use AJAX techniques on their websites and applications? What's AJAX? It's a description for functionality that uses a JavaScript feature called `XmlHttpRequest` to make requests to the server in the background and use the results within the page.

The Web is based around the concept of pages. When you click a link or submit a form, usually you leave the page that you're on and go to a new page with a different URL (or refresh the current page). Frames added the ability to split a page into sections and refresh each section independently. So, you could click a link in a navigation frame and reload the main content while leaving the other sections alone. AJAX is about retrieving content from the server and then placing it on the page using JavaScript.

In the previous example, you saw how you can enter information in a form and add it to the current page. Using AJAX, you can use the same techniques to retrieve data from the server and add it to the page. It's possible to write the code necessary to do this sort of thing from scratch, but jQuery and other libraries make it a whole lot easier.

Usually, AJAX is associated with server-side applications. For example, you can create a search engine and then use AJAX to retrieve search results and present them without ever leaving the current page. Unfortunately, there's not enough space in this book to teach you how to create a search engine. jQuery provides the ability to retrieve information from a different static page using AJAX and present it in the current page. I'm going to present an example that takes advantage of that feature to show you how AJAX can be used.

Using AJAX to Load External Data

I've created a simple page, shown in Figure 16.16, that allows users to look up information about South American countries. When a user clicks one of the links, the information about that country is retrieved from the server and displayed inline on the page.

NOTE

Because of the way AJAX works, this example will work only if it's deployed on a web server. If you load the files directly in your browser, the JavaScript code that retrieves the information won't work.

The sample consists of two files. The first is the page shown in Figure 16.16, which loads the data from the second page.

FIGURE 16.16
A page that loads data from an external source using AJAX.

The second is the page containing the information about the countries. Here's the source for the second page, `countries.html`, which contains information about the countries:

```
<!DOCTYPE html>
<html>
<head>
        <title>South American Countries</title>
</head>
<body>
<div id="uruguay">
  <h2>Uruguay</h2>

  <p>Uruguay, officially the Oriental Republic of Uruguay, is a country located
in the southeastern part of South America. It is home to some 3.5 million peo-
ple, of whom 1.4 million live in the capital Montevideo and its metropolitan
area. An estimated 88% of the population are of European descent.</p>
  <p>Uruguay's only land border is with Rio Grande do Sul, Brazil, to the north.
To the west lies the Uruguay River, to the southwest lies the estuary of Rio de
la Plata, with Argentina only a short commute across the banks of either of
these bodies of water, while to the southeast lies the southern part of the
Atlantic Ocean. Uruguay, with an area of approximately 176 thousand km2, is the
second smallest nation of South America in area after Suriname.</p>
</div>

<div id="paraguay">
  <h2>Paraguay</h2>
  <p>Paraguay, officially the Republic of Paraguay, is a landlocked country in
South America. It is bordered by Argentina to the south and southwest, Brazil to
the east and northeast, and Bolivia to the northwest. Paraguay lies on both
banks of the Paraguay River, which runs through the center of the country from
north to south. Because of its central location in South America, is sometimes
referred to as Corazon de America, or the Heart of America.</p>
  <p>As of 2009 the population was estimated at 6.3 million. The capital and
largest city is Asuncion. The official languages are Spanish and Guarani, both
being widely spoken in the country. Most of the population are mestizos.</p>
</div>
</body>
</html>
```

For a real application, instead of this simple page, you'd have a more robust service that could return lots of information about every country in South America on demand. This example page illustrates the concept without requiring any knowledge of server-side programming.

Now that the raw information is in place to be used on the page, I'll explain how the page works. When a link is clicked, the information is retrieved from `countries.html` and displayed on the initial page, as shown in Figure 16.17.

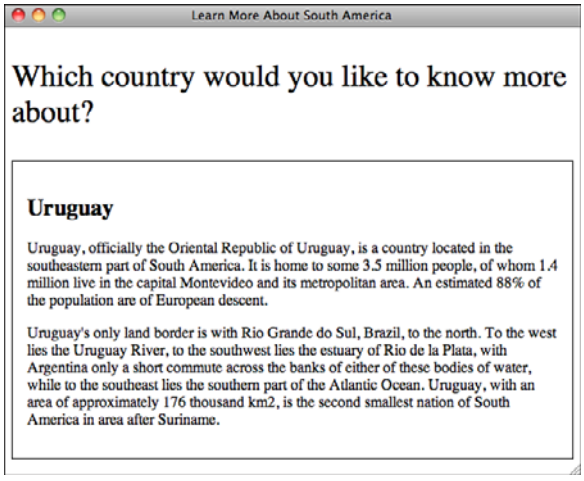


FIGURE 16.17
The information about Uruguay was loaded from an external source.

First, let's look at the two links:

```
<p class="countryOption"><a href="countries.html #uruguay">Uruguay</a></p>
<p class="countryOption"><a href="countries.html #paraguay">Paraguay</a></p>
```

They almost look like regular links. The one difference is that I've included a space between the filename and the anchor in the URL. That's because it's not actually an anchor, it's the ID of a `<div>` on the `countries.html` page.

Here's the event handler for the `click` event for the links:

```
$("p.countryOption a").click(function (event) {
  event.preventDefault();
  $("p.countryOption").fadeOut();
  $("#country").load($(this).attr('href'));
  $("#country").fadeIn();
});
```

You should be used to most of this by now. The first line prevents the link from actually taking you to the link referenced in the `href` attribute. The second line fades out the links, because they'll be replaced by the country data.

The third line actually performs the AJAX request. It instructs jQuery to load whatever is in the `href` of the link the user clicked on into the element with the ID "country." In this case, the links refer to jQuery selectors of sorts. Remember the URLs in the links? They consist of two parts, the first being the file to load, and the second being a jQuery selector, in this case, the ID of the country that I'll be displaying information about. jQuery loads the entire page and then applies the selector to it to extract the information I care about.

Here's the full source code for the page:

```
<!DOCTYPE html>

<html>
<head>
       <title>Learn More About South America</title>
       <script src="jquery-1.4.2.min.js" type="text/javascript" charset="utf-
8"></script>
       <script type="text/javascript" charset="utf-8">
       $(function () {
         $("#country").hide();

         $("p.countryOption a").click(function (event) {
           event.preventDefault();
           $("p.countryOption").fadeOut();
           $("#country").load($(this).attr('href'));
           $("#country").fadeIn();
         });
  });
       </script>
       <style type="text/css" media="screen">
         #country { border: 1px solid black; padding: 15px; }
         p.question { font-size: 200%; }
       </style>
</head>
<body>

<p class="question">
  Which country would you like to know more about?
</p>

<div id="country">Foo</div>

<p class="countryOption"><a href="countries.html #uruguay">Uruguay</a></p>
<p class="countryOption"><a href="countries.html #paraguay">Paraguay</a></p>

</body>
</html>
```

To read about other AJAX-related methods offered by jQuery, take a look at the jQuery API documentation. Most of the other jQuery methods are more suitable to application development, but they essentially work in a similar fashion to the `load()` method that you saw here.

Summary

In this lesson, I explored some of the powerful features common to most JavaScript libraries using jQuery. You learned which JavaScript libraries are available and why you might want to use them. You also learned how to include jQuery in a web page and take advantage of its functionality through the `document.ready()` event. I explained how event binding works with jQuery and how to dynamically modify the styles on a page as well as the content of a page itself. Finally, I explained what AJAX is and how jQuery and other JavaScript libraries enable you to make requests to external data sources from within a web page.

Workshop

As always, we wrap up the lesson with a few questions, quizzes, and exercises. Here are some questions and exercises that should refresh what you've learned about JavaScript libraries and jQuery.

Q&A

Q Won't adding a JavaScript library cause my pages to load more slowly?

A Yes, adding a JavaScript library will add to your overall page size. However, the browser will cache the external JavaScript file, so it should only have to download it once, when they get to the first page of your site. When they go to subsequent pages, the JavaScript library will already be in the cache. Also, the libraries vary in size. If you are concerned about download time, you might want to go with a smaller library.

Q What about users who don't have JavaScript enabled?

A It's generally agreed that less than 5 percent of users have JavaScript disabled these days. However, you'll still want to make sure that essential functionality still works for users who don't have JavaScript access. That's one of the big advantages of the unobtrusive JavaScript approach that these libraries reinforce. The markup should work fine without the JavaScript, which enhances the experience but is not essential to making the pages work.

Quiz

1. What's the basic function of a web server?
2. How can you obtain an Internet connection?
3. What are default index files, and what's the advantage of using them in all directories?
4. What are some things that you should check immediately after you upload your web pages?
5. Name some of the ways that you can promote your website.
6. What's a hit?
7. What are the advantages of using an all-in-one submission page to promote your site?

Quiz Answers

1. A web server is a program that sits on a machine on the Internet (or an intranet). It determines where to find files on a website and keeps track of where those files are going.
2. You can obtain an Internet connection through school, work, or commercial Internet or web services, or you can set up your own web server.
3. The default index file is loaded when a URL ends with a directory name rather than a filename. Typical examples of default index files are `index.html`, `index.htm`, and `default.htm`. If you use default filenames, you can use a URL such as http://www.mysite.com/ rather than http://www.mysite.com/index.html to get to the home page in the directory.
4. Make sure that your browser can reach your web pages on the server, that you can access the files on your website, and that your links and images work as expected. After you've determined that everything appears the way you think it should, have your friends and family test your pages in other browsers.
5. Some ways you can promote your site include major web directories and search engines, listings on business cards and other promotional materials, and web rings.
6. A hit is a request for any file from your website.
7. An all-in-one submission page enables you to submit your URL to several different site promotion areas and web robots simultaneously. Some provide a small number of submissions for free and a larger number of submissions for an additional fee.

Exercises

1. Download jQuery and use the `<script>` tag to load it in a web page.
2. Use jQuery to disable all the links on a web page.
3. Use jQuery to cause a border to appear around all the links on a web page when the user mouses over them. Make sure to remove the borders when the user moves the pointer away from the link.
4. Try to add a link to a web page that uses AJAX to retrieve the local temperature from a weather site for your city. You'll need to find the URL for a page with your city's weather on it, and then create the correct selector to extract only the information you want from that page. After you find the weather page, view the source on it to figure out how to extract the information using a jQuery selector.

This page intentionally left blank

LESSON 17

Working with Frames and Linked Windows

In the early days of the Web, two significant limitations of web browsers were that they could only display one document in a browser window at a time and that sites couldn't open more browser windows if needed. Frames enable you to divide the browser window into sections, each containing a different document, and linked windows enable you to create links that open pages in new browser windows. Used properly, these two techniques can make your website easier to navigate. Unfortunately, they can also be misused to make your site confusing, difficult to use, and annoying to your users. In this lesson, you'll learn how to apply these techniques to make your site more usable.

You'll learn all about the following topics:

- What frames are, how they can affect your layout, and who supports them
- How to work with linked windows
- How to work with frames
- How to create complex framesets
- Inline frames

What Are Frames?

In this lesson, you'll learn about the tags that you can use to create frames. There are two ways to use frames. The first is to divide the page into sections using a frameset and then to load documents into each of the frames, and the second is to embed a second document into an existing page using an inline frame. Inline frames are like images or Flash movies, they can be included anywhere within a page. Over time, inline frames have become more commonly used, and frames created using framesets have become less commonly used. HTML5 does not include support for frames created using framesets, leaving only inline frames. Browser support for framesets is still all but universal, but for new projects, you should probably avoid using framesets whenever possible.

Frames give you an entirely different level of layout control than you've had so far in this book. For example, take a look at the example in Figure 17.1.

FIGURE 17.1
A sample web page with frames.

On this screen, you see four frames displayed within one browser window. The left frame contains graphical navigation elements, the lower-right frame contains text navigational links, the upper-right frame contains the page header, and the middle-right frame contains the main content of the site. This screenshot also illustrates one of the disadvantages of using frames. The frame that contains the actual page content actually uses a fairly small section of the browser window; the rest of the window is consumed by the other frames. When you separate your layout using frames, you can detract from the important content on your site.

Because the information displayed on the page is separated into individual frames, the contents of a single frame can be updated without affecting the contents of any other frame. If you click one of the linked images in the left frame, for example, the contents of the large frame on the right are automatically updated to display the details about the subject you've selected. When this update occurs, the contents of the left frame and the bottom frame aren't affected.

Working with Linked Windows

Before you learn how to use frames, you need to learn about the `target` attribute of the `<a>` tag. It enables you to direct a link to open a page in a different frame or window, and learning how to use it is necessary to build sites with frames that work. This attribute takes the following form:

```
target="window_name"
```


Usually, when you click a hyperlink, the page to which you're linking replaces the current page in the browser window. When you use the `target` attribute, you can open links in new windows, or in existing windows other than the one that the link is in. With frames, you can use the `target` attribute to display the linked page in a different frame. The `target` attribute tells the web browser to display the information pointed to by a hyperlink in a window called *window_name*. Basically, you can call the new window anything you want, except that you can't use names that start with an underscore (_). These names are reserved for a set of special `target` values that you'll learn about later in the "Magic `target` Names" section.

When you use the `target` attribute inside an `<a>` tag, the browser first checks whether a window with the name *window_name* exists. If it does, the document pointed to by the hyperlink replaces the current contents of *window_name*. On the other hand, if no window called *window_name* currently exists, a new browser window opens with that name assigned to it. Then the document pointed to by the hyperlink is loaded into the newly created window.

Task: Exercise 17.1: Working with Windows

Framesets rely on the `target` attribute to load pages into specific frames in a frameset. Each of the hyperlinks in the following exercise uses the `target` attribute to open a web page in a different browser window. The concepts you'll learn here will help you understand later how targeted hyperlinks work in a frameset.

In this exercise, you create four separate HTML documents that use hyperlinks, including the `target` attribute. You use these hyperlinks to open two new windows called `yellow_page` and `blue_page`, as shown in Figure 17.2. The top window is the original web browser window (the red page), `yellow_page` is at the bottom left, and `blue_page` is at the bottom right.

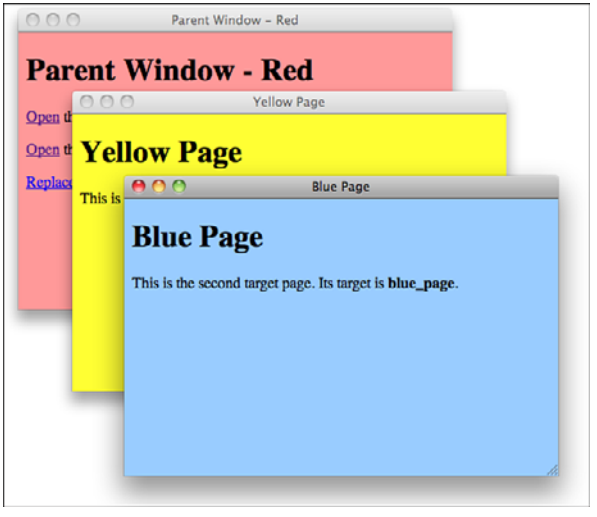


FIGURE 17.2
Using the *target* attribute indicates that links should open new windows.

First, create the document to be displayed by the main web browser window, shown in Figure 17.3, by opening your text editor of choice and entering the following lines of code:

Input

```
<!DOCTYPE html>
<html>
<head>
<title>Parent Window - Red</title>
<style type="text/css" media="screen">
    body {
        background-color: #ff9999;
    }
</style>
</head>
<body>
    <h1>Parent Window - Red</h1>

    <p><a href="yellow.html" target="yellow_page">Open</a> the Yellow Page in a
new window.</p>
```

```
    <p><a href="blue.html" target="blue_page">Open</a> the Blue Page in a new
window.</p>

    <p><a href="green.html" target="yellow_page">Replace</a> the yellow page
with the Green Page.</p>
</body>
</html>
```

Output ▶

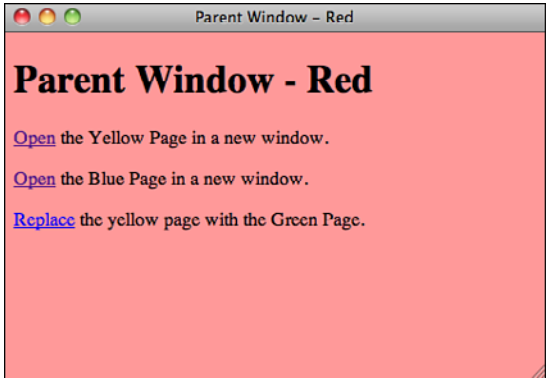


FIGURE 17.3
The parent window (the red page).

This creates a light-red page that links to the other three pages. Save this HTML source as `parent.html`.

Next, create a document called `yellow.html` (see Figure 17.4) by entering the following code:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Yellow Page</title>
<style type="text/css" media="screen">
    body {
        background-color: #FFFF33;
    }
</style>
</head>
<body>
    <h1>Yellow Page</h1>

    <p>This is the first target page. Its target is <b>yellow_page</b>.</p>
</body>
</html>
```

Output ▶

FIGURE 17.4
yellow.html web browser window named *yellow_page*.

After saving `yellow.html`, create another document called `blue.html` (see Figure 17.5) by entering the following code:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Blue Page</title>
<style type="text/css" media="screen">
    body {
        background-color: #99ccff;
    }
</style>
</head>
<body>
    <h1>Blue Page</h1>

    <p>This is the second target page. Its target is <b>blue_page</b>.</p>
</body>
</html>
```

Output ▶

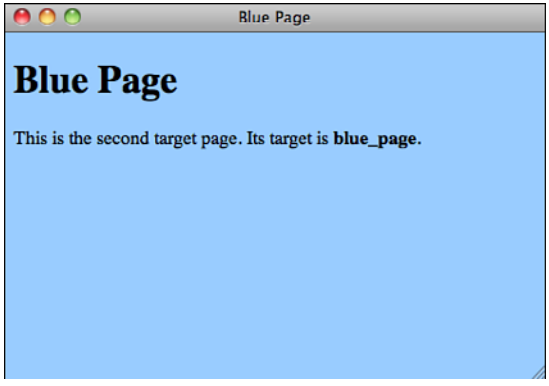


FIGURE 17.5
blue.html displayed in the web browser window named *blue_window*.

Next, create a fourth document called `green.html`, which looks like the following:

```
<!DOCTYPE html>
<html>
<head>
<title>Green Page</title>
<style type="text/css" media="screen">
    body {
        background-color: #ccffcc;
    }
</style>
</head>
<body>
    <h1>Green Page</h1>

    <p>This is the third target page. Its target is <b>yellow_page</b>. It
should
    replace the yellow page in the browser.</p>
</body>
</html>
```

To complete the exercise, load `parent.html` (the red page) into your web browser. Click the first hyperlink to open the yellow page in a second browser window. This happens because the first hyperlink contains the attribute `target="yellow_page"`, as the following code from `parent.html` demonstrates:

```
<p><a href="yellow.html" target="yellow_page">Open</a> the Yellow Page in a
 new window.<br />
```

▼ Now return to the red page and click the second link. The blue page opens in a third browser window. Note that the new windows probably won't be laid out like the ones shown in Figure 17.2; they usually overlap each other. The following `target="blue_page"` statement in the `parent.html` page is what causes the new window to open:

```
<a href="blue.html" target="blue_page">Open</a> the Blue Page in a new
 window.</p>
```

The previous two examples opened each of the web pages in a new browser window. The third link, however, uses the `target="yellow_page"` statement to open the green page in the window named `yellow_page`. You accomplish this using the following code in `parent.html`:

```
<p><a href="green.html" target="yellow_page">Replace</a> the yellow page
 with the Green Page.</p>
```

Because you already opened the `yellow_page` window when you clicked the link for the yellow page, the green page should replace the page that's already in it. To verify this, click the third hyperlink on the red page. This replaces the contents of the yellow page (with the `yellow_page` target name) with the green page (`green.html`), as shown in Figure 17.6.

FIGURE 17.6 *green.html* displayed in the web browser window named *green_page*.

▲

The <base> Tag

When you're using the `target` attribute with links, you'll sometimes find that all or most of the hyperlinks on a web page should point to the same window. This is especially true when you're using frames, as you'll discover in the following section.

In such cases, instead of including a `target` attribute for each `<a>` tag, you can use another tag, `<base>`, to define a global target for all the links on a web page. The `<base>` tag is used as follows:

```
<base target="window_name">
```

If you include the `<base>` tag in the `<head>...</head>` block of a document, every `<a>` tag that doesn't have a `target` attribute will be directed to the window indicated by the `base` tag. For example, if you had included the tag `<base target="yellow_page">` in the HTML source for `parent.html`, the three hyperlinks could have been written as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Parent Window - Red</title>
<style type="text/css" media="screen">
    body {
        background-color: #ff9999;
    }
</style>
<base target="yellow_page"> <!— add base target="value" here —>
</head>
<body>
    <h1>Parent Window - Red</h1>

    <p><a href="yellow.html" target="yellow_page">Open</a>
    <!— no need to include a target —>
         the Yellow Page in a new window.</p>

    <p><a href="blue.html" target="blue_page">Open</a> the Blue Page in a new
window.</p>

    <p><a href="green.html" target="yellow_page">Replace</a>
    <!— no need to include a target —>
         the yellow page with the Green Page.</p>
</body>
</html>
```

In this case, `yellow.html` and `green.html` load into the default window assigned by the `<base>` tag (`yellow_page`); `blue.html` overrides the default by defining its own target window of `blue_page`.

You also can override the window assigned with the `<base>` tag by using one of two special window names. If you use `target="_blank"` in a hyperlink, it opens a new browser window that doesn't have a name associated with it. Alternatively, if you use `target="_self"`, the current window is used rather than the one defined by the `<base>` tag.

NOTE

If you don't provide a `target` using the `<base>` tag and you don't indicate a target in a link's `<a>` tag, the link will load the new document in the same frame or window as the link.

Working with Frames

Frames make it possible to split a single web "page" into multiple independent parts, each of which can display a document on its own. When frames were originally introduced, this was a much bigger deal than it is today. One of the key benefits of frames is that they enable you to make part of a site visible all the time even while other parts can be scrolled out of view. This can now be done using CSS positioning. Frames enable you to load new content into one section of a layout while leaving the rest of the page in place, but that can be accomplished using JavaScript.

That is not to say that there's no place for the use of frames any more, but they are much less popular than they were in their heyday. There are some situations where frames are absolutely the best solution to the problem at hand, but these days there are a lot of solid alternatives, too.

Working with frames involves describing the frame layout and then choosing which documents will be displayed in each frame. Figure 17.7 shows how five separate documents are needed to create the screen shown earlier in Figure 17.1.

The first HTML document you need to create is called the *frameset document*. In this document, you define the layout of your frames, and the locations of the documents to be initially loaded in each frame. The document in Figure 17.7 has three frames.

Each of the three HTML documents other than the frameset document, the ones that load in the frames, contain normal HTML tags that define the contents of each frame. These documents are referenced by the frameset document.

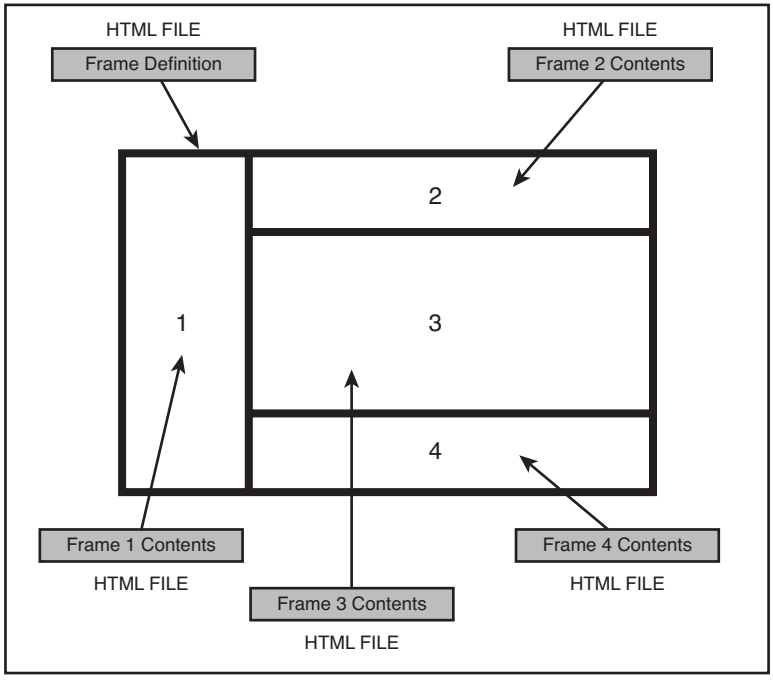


FIGURE 17.7
You must create a separate HTML document for each frame.

The <frameset> Tag

To create a frameset document, you begin with the <frameset> tag. When used in an HTML document, the <frameset> tag replaces the <body> tag, as shown in the following code:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<frameset>
  .. your frameset goes here ...
</frameset>
</html>
```

It's important that you understand upfront how a frameset document differs from a normal HTML document. If you include a <frameset> tag in an HTML document, you cannot include a <body> tag also. The two tags are mutually exclusive. In addition, no other formatting tags, hyperlinks, or document text should be included in a frameset document. (The exception to this is the <noframes> tag, which you learn about later in this lesson in

the section called, appropriately enough, "The <noframes> Tag.") The <frameset> tags contain only the definitions for the frames in this document—what's called the page's *frameset*.

The HTML 4.01 specification supports the <frameset> tag along with two possible attributes: cols and rows.

The cols Attribute

When you define a <frameset> tag, you must include one of two attributes as part of the tag definition. The first of these attributes is the cols attribute, which takes the following form:

```
<frameset cols="column width, column width, ...">
```

The cols attribute tells the browser to split the screen into a number of vertical frames whose widths are defined by *column width* values separated by commas. You define the width of each frame in one of three ways: explicitly in pixels, as a percentage of the total width of the <frameset>, or with an asterisk (*). When you use the asterisk, the frames-compatible browser uses as much space as possible for that frame.

When included in a complete frame definition, the following <frameset> tag splits the browser into three vertical frames, as shown in Figure 17.8. The fifth line in the following code example creates a left frame 100 pixels wide, a middle column that's 50% of the width of the browser, and a right column that uses all the remaining space:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Three Columns</title>
</head>
<frameset cols="100,50%,*">
  <frame src="leftcol.html">
  <frame src="midcol.html">
  <frame src="rightcol.html">
</frameset>
</html>
```

Output ▶

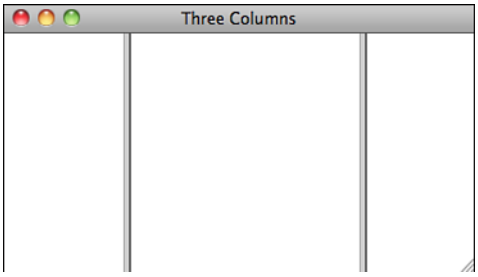


FIGURE 17.8
The *cols* attribute defines the number of vertical frames or columns in a frameset.

NOTE

Because you're designing web pages for users with various screen sizes, you should use absolute frame sizes sparingly. Whenever you do use an absolute size, ensure that one of the other frames is defined using an * to take up all the remaining screen space.

TIP

To define a frameset with three columns of equal width, use `cols="*,*,*"`. This way, you won't have to mess around with percentages because frames-compatible browsers automatically assign an equal amount of space to each frame assigned a width of *.

The `rows` Attribute

The `rows` attribute works the same as the `cols` attribute, except that it splits the screen into horizontal frames rather than vertical ones. To split the screen into two frames of equal height, as shown in Figure 17.9, you would write the following:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Two Rows</title>
</head>
<frameset rows="50%,50%">
  <frame src="toprow.html">
  <frame src="bottomrow.html">
</frameset>
</html>
```

Alternatively, you could use the following line:

```
<frameset rows="*,*">
```

Output ▶

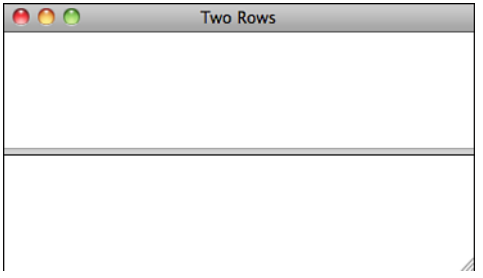


FIGURE 17.9
The *rows* attribute defines the number of horizontal frames or rows in a frameset.

NOTE

If you try either of the preceding examples, you'll find that the `<frameset>` tag doesn't appear to work. You get this result because there are no contents defined for the rows or columns in the frameset. To define the contents, you need to use the `<frame>` tag, which is discussed in the next section.

The `<frame>` Tag

After you have your basic frameset laid out, you need to associate an HTML document with each frame using the `<frame>` tag, which takes the following form:

```
<frame src="document URL">
```

For each frame defined in the `<frameset>` tag, you must include a corresponding `<frame>` tag, as shown in the following:

Input ▼

```
<!DOCTYPE html>
<html>
        <head>
                <title>The FRAME Tag</title>
        </head>
        <frameset rows="*,*,*">
                <frame src="document1.html">
                <frame src="document2.html">
                <frame src="document3.html">
        </frameset>
</html>
```

This example defines a frameset with three horizontal frames of equal height (see Figure 17.10). The contents of `document1.html` are displayed in the first frame, the contents of `document2.html` in the second frame, and the contents of `document3.html` in the third frame.

Output ▶

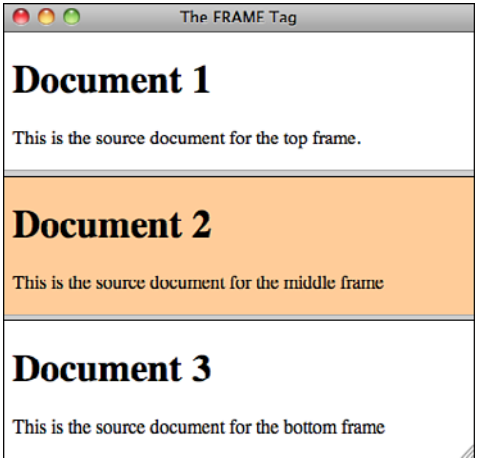


FIGURE 17.10
You use the *<frame>* tag to define the contents of each frame.

TIP

When you're creating frameset documents, you might find it helpful to indent the `<frame>` tags so that they're separated from the `<frameset>` tags in your HTML document. This has no effect on the appearance of the resulting web pages, but it does tend to make the HTML source easier to read.

The `<noframes>` Tag

What happens if a browser that doesn't support frames navigates to a frameset document? Nothing. You get a blank page. Fortunately, there's a way around this problem.

A special tag block called `<noframes>` enables you to include additional HTML code as part of the frameset document. The code you enclose within the `<noframes>` element isn't displayed in frames-compatible browsers, but it's displayed in browsers that don't support frames. The `<noframes>` tag takes the following form:

```
<!DOCTYPE html>
<html>
<head>
<title>Frameset with No Frames Content</title>
</head>
```

```
<frameset>
 your frameset goes here.
<noframes>
 Include any text, hyperlinks, and tags you want to here.
</noframes>
</frameset>
</html>
```

Using the frames' content and tags inside `<noframes>`, you can create pages that work well with both kinds of browsers. Later in this lesson, you add some `<noframes>` content to a frameset.

NOTE

The way the `<noframes>` tag works is actually kind of interesting. It works because web browsers are designed to ignore tags that they don't understand. So, browsers that don't support frames ignore the `<frameset>` and `<frame>` tags. They also ignore the `<noframes>` tag and just display whatever is inside it. Browsers that do support frames know to render the frames and ignore the text inside the `<noframes>` tag.

You might wonder which browsers lack frame support these days. The answer is that browsers on mobile browsers often do not support them. These days you should take mobile devices into account when you create websites, because their use is growing rapidly.

Changing Frame Borders

Notice that all the frames in this lesson have thick borders separating them. A number of attributes can be set to control the appearance of frame borders or prevent them from appearing altogether.

Start with the `<frame>` tag. The `frameborder` attribute is used to enable or disable the default, browser-generated border between frames, which is usually gray and has a beveled edge. `frameborder` takes two possible values: `1` (to display borders) or `0` (to turn off the display of borders).

Frames can be styled using *Cascading Style Sheets (CSS)*, so if you want to add your own border to a frame, disable the browser-supplied borders using the `frameborder` attribute and add your own using the CSS border property.

NOTE

If you turn off the border, frames-compatible browsers won't display its default three-dimensional border. However, a space will still be left for the border.

For example, the following code adds a deep red border around the middle frame in the frameset using CSS:

```
<!DOCTYPE html>
<html>
        <head>
                <title>The FRAME Tag</title>
                <style type="text/css" media="screen">
                        frame.middle {
                                border-top: 3px solid #cc3333;
                                border-bottom: 3px solid #cc3333;
                        }
                </style>
        </head>
        <frameset rows="*,*,*">
                <frame frameborder="0" src="document1.html">
                <frame frameborder="0" class="middle" src="document2.html">
                <frame frameborder="0" src="document3.html">
        </frameset>
</html>
```


As you can see, you can use classes (and IDs) with frames just as you can with other HTML elements. Of course, there's room for confusion when borders are defined. In the following frameset definition, a conflict arises because the top frame has the frame border enabled, but the middle frame does not. In this case, if either frame has the border turned on, it is displayed.

```
<!DOCTYPE html>
<html>
        <head>
                <title>The FRAME Tag</title>
        </head>
        <frameset rows="*,*,*">
                <frame frameborder="1" src="document1.html">
                <frame frameborder="0" src="document2.html">
                                        <frame frameborder="0"
src="document3.html">
        </frameset>
</html>
```

In addition, the frameset is defined as having no borders, but the first frame is supposed to have a border. How do you resolve this problem? You can apply three simple rules:

- Attributes in the outermost frameset have the lowest priority.
- Attributes are overridden by attributes in a nested `<frameset>` tag.
- Any `bordercolor` attribute in the current frame overrides previous ones in `<frameset>` tags.

Additional Attributes

Table 17.1 shows a few extra attributes for the `<frame>` tag. These attributes can give you additional control over how the user interacts with your frames. Other attributes control margins or spacing between frames and whether scrollbars appear when required.

TABLE 17.1 Control Attributes for the `<frame>` Tag

Attribute	Value	Description
`frameborder`	`1`	Displays borders around each frame (default).
`frameborder`	`0`	Creates borderless frames.
`longdesc`	*URL*	Specifies a URL that provides a longer description of the contents of the frameset. Primarily used with nonvisual browsers.
`marginheight`	*pixels*	To adjust the margin that appears above and below a document within a frame, set `marginheight` to the number indicated by *pixels*.
`marginwidth`	*pixels*	The `marginwidth` attribute enables you to adjust the margin on the left and right sides of a frame to the number indicated by *pixels*.
`name`	*string*	Assigns a name to the frame for targeting purposes.
`noresize`		By default, the users can move the position of borders around each frame on the current screen by grabbing the borders and moving them with the mouse. To lock the borders of a frame and prevent them from being moved, use the `noresize` attribute.
`scrolling`	`auto`	(Default) If the content of a frame takes up more space than the area available, frames-compatible browsers automatically add scrollbars to either the right side or the bottom of the frame so that the users can scroll through the document.

TABLE 17.1 Continued

Attribute	Value	Description
`scrolling`	`no`	Setting the value of `scrolling` to `no` disables the use of scrollbars for the current frame. (Note that if you do this but the document contains more text than can fit inside the frame, users can't scroll the additional text into view.)
`scrolling`	`yes`	If you set `scrolling` to `yes`, scrollbars are included in the frame even if they aren't required.
`src`	*URL*	Specifies the URL of the initial source document that appears in a frame when the frameset first opens in the browser.

Creating Complex Framesets

The framesets you've learned about so far are the most basic types of frames that can be displayed. In day-to-day use, however, you'll rarely use these basic frame designs. On all but the simplest sites, you'll most likely want to use more complex framesets.

Therefore, to help you understand the possible combinations of frames, links, images, and documents that can be used by a website, this section explores complex framesets.

Task: Exercise 17.2: Creating the Content Pages for Your Frameset

Most commonly, framesets are used to keep navigational elements in view as the user scrolls through the contents of the document. Each time the visitor clicks a link in the left navigation frame, the content in the main frame displays the selected page. The (very silly) frameset that you'll create in this exercise demonstrates this technique. Although it's not a practical example, it's simple, fun, and demonstrates the same techniques you would use for a navigation bar.

When you design a web page that uses frames, you normally design the frameset before you go through all the trouble of designing the content that goes into it. That's because you'll want to know how big your frames are going to be before you start designing graphics and other page content to put into them.

I'm doing things a little backward here, but for good reason. It might help you to better understand how things fit together if you see real content in the frames as you design the frameset. For this reason, I have you design the content first.

The following content pages don't include any of the frameset tags discussed so far. There are eight pages in all, but I promise that I keep the code for these pages brief. Ready?

TIP

Sometimes you may want to work on your frame document before you've finished all the pages that will be displayed in the frames. However, your frameset won't be displayed properly when it's loaded into a frames-compatible browser for testing unless you define `<frame>` tags that include valid documents. If you want to design a frameset before you create the content, you can create a small empty HTML document called `dummy.html` and use it for all your frame testing.

The frameset that you'll create in Exercises 17.3 through 17.7 consists of three frames. The layout of the frameset will be as shown in Figure 17.11. The frameset page loads first and instructs the browser to divide the browser window into three frames. Next, it loads the three pages that appear in the top, left, and main frames. Finally, if a user browses to the frameset without a frames-compatible browser, an alternative page will appear.

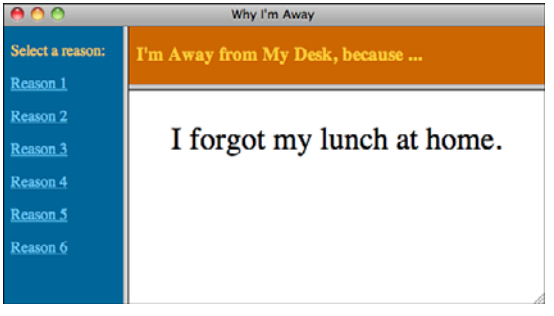


FIGURE 17.11
You'll create a frameset that consists of three frames: top, left, and main.

The top frame always displays the same web page: `away.html`. The `choices.html` page that appears in the frame on the left side contains a list of links to six different pages named `reason1.html` through `reason6.html`. Each of these six pages will load into the main frame on the bottom-right portion of the frameset.

Start with the page displayed in the top frame. This page will always appear in the frameset. Here you can include any information you want to display permanently as visitors browse through your site. Real-world examples for the content of this frame include the name of your website, a site logo, a link to your email address, or other similar content. Type in the following code and save it to your hard drive as `away.html`:

```
<!DOCTYPE html>
<html>
        <head>
                <title>I'm Away from My Desk Because</title>
                <style type="text/css" media="screen">
                        body { background-color: #cc6600; color: #ffcc33; }
                </style>
        </head>
        <body>
                <h3>I'm Away from My Desk, because ...</h3>
        </body>
</html>
```


Figure 17.12 shows this page.

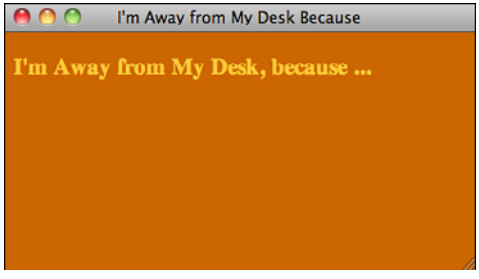


FIGURE 17.12
The top frame in the frameset.

Next, you'll create the left frame in the frameset. On real websites, this is typically the frame used for text or image navigation links that take your visitors to several different key pages on your site. For example, a personal site might have a navigation bar that takes its visitors to a home page, a guest book, a links page, and other sections of interest. A corporate or business site could contain links for products, customer support, frequently asked questions, employment opportunities, and so on.

The contents page in the following example works exactly the same way that a real-world navigation bar does. When the appropriate link is selected, it displays one of the six pages in the main frame of the frameset. The contents page contains links to six pages, `reason1.html` through `reason6.html`, which you'll create next.

After you enter the following code into a new page, save it to your hard drive in the same directory as the first page and name it `choice.html`:

Input

```
<!DOCTYPE html>
<html>
        <head>
                <title>Reason I'm Out</title>
                <style type="text/css" media="screen">
                        body {
                                color: #ffcc66;
                                background-color: #006699;
                        }

                        a { color: #ffffff; }
                        a:visited { color: #66ccff; }
                        a:active { color: #ff6666; }
                </style>
        </head>
        <body>
                <p>Select a reason:</p>
                <p><a href="reason1.html">Reason 1</a></p>
                <p><a href="reason2.html">Reason 2</a></p>
                <p><a href="reason3.html">Reason 3</a></p>
                <p><a href="reason4.html">Reason 4</a></p>
                <p><a href="reason5.html">Reason 5</a></p>
                <p><a href="reason6.html">Reason 6</a></p>
        </body>
</html>
```

Your page should look as shown in Figure 17.13 when you open it in a browser.

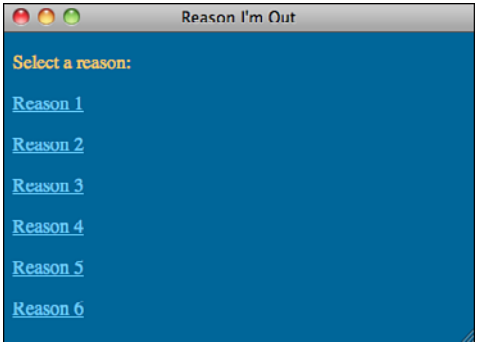


FIGURE 17.13 The left frame in the frameset.

Now you need to create the six pages that will appear in the main frame when the visitor selects one of the links in the contents frame. The main frame is designed to display pages that normally you would display in a full browser window. However, if you're going to display your pages in a frameset that has a left navigation bar, you'll have to account for the reduced size of the frame in your design.

To keep the page examples relatively easy, I've given them all the same basic appearance. This means that the code for all of these pages is pretty much the same. The only items that change from page to page are the following:

- The title of the page
- The description of my current mood
- The text that describes what each image means

To create the first of the six pages that will appear in the main frame, type the following code into a new page and save it as `reason1.html`:

Input ▼

```
<!DOCTYPE html>
<html>
        <head>
                <title>Reason 1 - Forgot My Lunch</title>
                <style type="text/css" media="screen">
                        body {
                                background-color: #ffffff;
                                font: 200% bold;
                                text-align: center;
                        }
                </style>
        </head>
        <body>
                <p>I forgot my lunch at home.</p>
        </body>
</html>
```

Figure 17.14 shows what this page should look like in a browser.

Output ▶

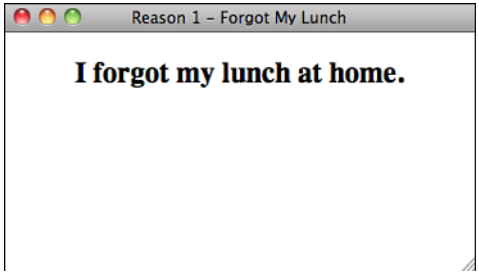


FIGURE 17.14
The first of the six pages that appear in the main frame.

▼ You code the remaining five pages for the main frame similarly. Modify the page you just created to build the second of the six main pages. The only differences from the previous code (`reason1.html`) are shown with a gray background. Save the new page as `reason2.html`. The complete code appears as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Reason 2 - By the Water Cooler</title>
<style type="text/css" media="screen">
    body {
        background-color: #ffffff;
        font: 200% bold;
        text-align: center;
    }
</style>
</head>
<body>
    <p>I'm flirting by the water cooler.</p>
</body>
</html>
```

For the third page, modify the code again and save it as `reason3.html`. The complete code appears as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Reason 3 - Don't Ask!</title>
<style type="text/css" media="screen">
    body {
        background-color: #ffffff;
        font: 200% bold;
        text-align: center;
    }
</style>
</head>
<body>
    <p>None of your business!</p>
</body>
</html>
```

Here's the fourth page (`reason4.html`):

```
<!DOCTYPE html>
<html>
<head>
<title>Reason 4 - Out to Lunch</title>
<style type="text/css" media="screen">
```

```
    body {
        background-color: #ffffff;
        font: 200% bold;
        text-align: center;
    }
</style>
</head>
<body>
    <p>I'm out to lunch.</p>
</body>
</html>
```

The fifth page (`reason5.html`) looks like the following:

```
<!DOCTYPE html>
<html>
<head>
<title>Reason 5 - Boss's Office</title>
<style type="text/css" media="screen">
    body {
        background-color: #ffffff;
        font: 200% bold;
        text-align: center;
    }
</style>
</head>
<body>
    <p>The boss called me in his office.<p>
</body>
</html>
```

The last main page (`reason6.html`) appears as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Reason 6 - I Don't Work Here Anymore</title>
<style type="text/css" media="screen">
    body {
        background-color: #ffffff;
        font: 200% bold;
        text-align: center;
    }
</style>
</head>
<body>
    <p>I just got fired.<p>
</body>
</html>
```

Now you have the six pages that will appear in the main frame of the frameset. You're finally ready to build the frameset.

Task: Exercise 17.3: Combining `rows` and `cols`

To remind you of the basic layout of the frameset that you'll create, Figure 17.15 is another look at the complete page. It provides a simple example of how you can combine framesets to create complex designs.

Output

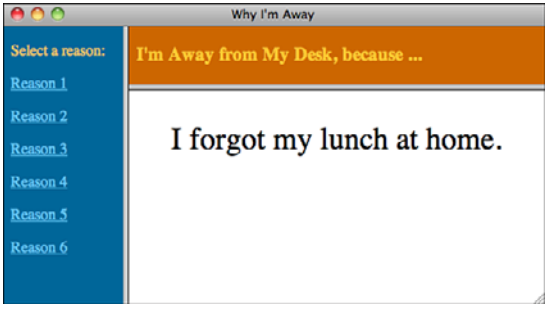


FIGURE 17.15
The frameset with three frames: top, left, and main.

TIP

When you're designing complex frame layouts, a storyboard is an invaluable tool. It helps you block out the structure of a frameset, and it can prove invaluable when you're adding hyperlinks (as you'll see in Exercise 17.5, "Using Named Frames and Hyperlinks").

In Figure 17.15, the right section of the screen is split into two horizontal frames, and the third frame at the left of the page spans the entire height of the screen. To create a frameset document that describes this layout, open your text editor and enter the following basic HTML structural details:

```
<!DOCTYPE html>
<html>
<head>
<title>Why I'm Away Frameset</title>
</head>
<frameset>
</frameset>
</html>
```

Next, you must decide whether you need to use a `rows` or `cols` attribute in your base `<frameset>`. Look at your storyboard—in this case Figure 17.15—and work out whether any frame areas extend right across the screen or from the top to the bottom. If any frames extend from the top to the bottom, as in this example, you need to start with a `cols` frameset; otherwise, you need to start with a `rows` frameset. On the other hand, if no frames extend completely across the screen either vertically or horizontally, you should start with a `cols` frameset.

To put it more simply, here are three easy-to-remember rules:

- Left to right, use `rows`
- Top to bottom, use `cols`
- Can't decide, use `cols`

NOTE

The reasoning behind the use of the "left to right, use `rows`" rule relates to how frames-compatible browsers create frames. Each separate `<frameset>` definition can split the screen (or a frame) either vertically or horizontally, but not both ways. For this reason, you need to define your framesets in a logical order to ensure that you achieve the layout you want.

In Figure 17.15, the left frame extends across the screen from top to bottom. As a result, you need to start with a `cols` frameset by using the rules mentioned previously. To define the base frameset, enter the following:

```
<frameset cols="125,*">
 <frame src="choice.html"> <!-- loads the choices page into the left frame -->
 <frame src="dummy.html">  <!-- this line is only temporary -->
</frameset>
```

Writing this code splits the screen into two sections. The first line defines a small frame at the left of the screen that is 125 pixels wide, and a large frame at the right of the screen that uses the rest of the available space.

As mentioned earlier in this lesson, the frameset document itself doesn't describe the contents of each frame. The documents specified in the `src` attribute of the `<frame>` actually contain the text, images, and tags displayed by the frameset. You can see an example of this tag in the second and third lines of the preceding code. The second line specifies the URL of the web page in the left frame (the `choice.html` page that you created earlier). The third line would display a web page named `dummy.html` (if you created one, that is), but we're just using this as a placeholder for the next exercise.

▼ Task: Exercise 17.4: Nesting Framesets

The next step in the process is to split the right frame area into two horizontal frames. You achieve this effect by placing a second `<frameset>` block inside the base `<frameset>` block. When one `<frameset>` block is nested inside another, the nested block must replace one of the `<frame>` tags in the outside frameset. In this case, you'll replace the line that loads the temporary `dummy.html` page (which doesn't really exist).

To split the right frame into two frame areas, you replace the dummy `<frame>` tag with an embedded `<frameset>` block. This embeds the new frameset inside the area defined for the `<frame>` tag it replaces. Inside the `<frameset>` tag for this new block, you need to define a `rows` attribute, as shown in the complete code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Why I'm Away</title>
  </head>
  <frameset cols="125,*">
  <!-- this loads the choices page into the left frame -->
    <frame src="choice.html">
      <frameset rows="60,*">          <!-- the frame for column 2 -->
        <frame src="away.html">       <!-- has been replaced -->
        <frame src="reason1.html">    <!-- with an embedded -->
      </frameset>                     <!-- frameset block -->
    </frameset>
</html>
```

The embedded `rows` frameset defines two rows, the first being 60% of the height of the embedded frame area and the second taking up all the remaining space in the embedded frame area. In addition, two `<frame>` tags are embedded inside the `<frameset>` block to define the contents of each column. The top frame loads `away.html`, and the bottom frame loads `reason1.html`.

NOTE

When used inside an embedded frameset, any percentage sizes are based on a percentage of the total area of the embedded frame, not on a percentage of the total screen.

▲ Save the finished HTML document to your hard drive as frameset.html. Test it in your browser.

Task: Exercise 17.5: Using Named Frames and Hyperlinks

If you were to load your `frameset.html` page into a frames-compatible browser at this stage, you would see a screen similar to the one shown in Figure 17.15. Some of the text sizes and spacing might differ slightly, but the general picture would be the same.

Although it looks right, it doesn't *work* right yet. If you click any of the hyperlinks in the left frame, the frames-compatible browser will attempt to load the contents of the file you select into the left frame. What you want it to do is to load each document into the larger right frame.

Earlier in this lesson, you learned about the `target` attribute, which loads different pages into a different browser window. To make the frameset work the way it should, you need to use a slight variation on the `target` attribute. Instead of the `target` pointing to a new window, you want it to point to one of the frames in the current frameset.

You can achieve this by first giving each frame in your frameset a frame name or window name. To do so, include a `name` attribute inside the `<frame>` tag, which takes the following form:

```
<frame src="document URL" name="frame name">
```

Therefore, to assign a name to each of the frames in the `frameset.html` document, you add the `name` attribute to each of the `<frame>` tags. Your frameset page now looks like the following, with the additions indicated with the shaded background:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Why I'm Away</title>
  </head>
  <frameset cols="125,*">
  <!-- this loads the choices page into the left frame -->
    <frame src="choice.html" name="left">
      <frameset rows="60,*">                    <!-- the frame for column 2 -->
        <frame src="away.html" name="top">      <!-- has been replaced -->
        <frame src="reason1.html" name="main"> <!-- with an embedded -->
      </frameset>                               <!-- frameset block -->
    </frameset>
</html>
```

This source code names the left frame `"left"`, the top-right frame `"top"`, and the bottom-right frame `"main"`. Next, resave the updated `frameset.html` file, and you're just about finished with the example.

▼ Task: Exercise 17.6: Linking Documents to Individual Frames

After you've named the frames, you have to fix the links in the `choice.html` page so that they load the target pages in the `main` frame rather than the `left` frame.

You might recall that the `target` attribute was used with the `<a>` tag to force a document to load into a specific window. You'll use the same attribute to control into which frame a document is loaded.

In this exercise, you want to load a page in the `main` (bottom-right) frame whenever you click a hyperlink in the `left` frame. Because you've already named the bottom-right frame `"main"`, all you need to do is add `target="main"` to each tag in the `choice.html` document. The following snippet of HTML source code demonstrates how to make this change:

```
<p><a href="reason1.html" target="main">Reason 1</a></p>
<p><a href="reason2.html" target="main">Reason 2</a></p>
<p><a href="reason3.html" target="main">Reason 3</a></p>
<p><a href="reason4.html" target="main">Reason 4</a></p>
<p><a href="reason5.html" target="main">Reason 5</a></p>
<p><a href="reason6.html" target="main">Reason 6</a></p>
```

Alternatively, you could use the `<base target="value">` tag because every tag in the `choice.html` document points to the same frame. In that case, you wouldn't need to include `target="main"` inside each `<a>` tag. Instead, place the following inside the `<head>...</head>` block of the document:

```
<base target="main">
```

With all the changes and new documents created, you should now load `frameset.html` into your frames-compatible browser and view all your HTML reference documents by selecting from the choices in the left frame.

TIP

After you get all your links working properly, you might need to go back and adjust the size of the rows and columns as defined in the `<frameset>` tags to get the layout exactly right. Remember, the final appearance of a frameset is still determined by the size of the screen and the visitor's operating system.

▲

Task: Exercise 17.7: Adding Your `noframes` Content

Although you have a frameset that works perfectly now, there's another feature you need to add to it. Remember, some people who visit your frames page won't be using frames-compatible browsers. The following addition to the frameset page creates some content that they'll see when they open the frameset.

Once again, open the `frameset.html` page. At this point, your code looks like the following:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Why I'm Away</title>
  </head>
  <frameset cols="125,*">
  <!— this loads the choices page into the left frame —>
    <frame src="choice.html" name="left">
      <frameset rows="60,*">                    <!-- the frame for column 2 -->
        <frame src="away.html" name="top">      <!-- has been replaced -->
        <frame src="reason1.html" name="main"> <!-- with an embedded -->
      </frameset>                               <!-- frameset block -->
    </frameset>
</html>
```

Immediately after the last `</frameset>` tag and before the final `</html>` tag, insert the following `<noframes>...</noframes>` element and content:

Input

```
<noframes>
  <body>
    <h1>I'm Away from My Desk, because ...</h1>
    <ul>
      <li>Reason 1 - <a href="reason1.html">I forgot my lunch at home.</a></li>
      <li>Reason 2 - <a href="reason2.html">I'm flirting by the water
cooler.</a></li>
      <li>Reason 3 - <a href="reason3.html">None of your business.</a></li>
      <li>Reason 4 - <a href="reason4.html">I'm out to lunch.</a></li>
      <li>Reason 5 - <a href="reason5.html">The boss just called me in his
office.</a></li>
      <li>Reason 6 - <a href="reason6.html">I just got fired.</a></li>
    </ul>
  </body>
</noframes>
```

When a user who isn't using a frames-compatible browser navigates to the frameset, she'll see the page that's similar to the one shown in Figure 17.16.

Output ▶

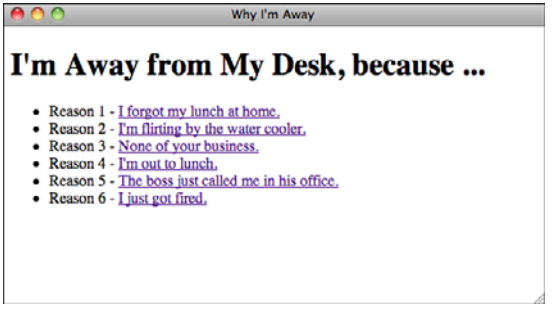


FIGURE 17.16
This page appears when users view the frameset with a browser that isn't frames-compatible.

▲

Magic `target` Names

Now that you've learned what the `target` attribute does in a frameset, you should know you can apply some special target names to a frameset.

You can assign four special values to a `target` attribute, two of which (`_blank` and `self`) you've already encountered. Netscape called these values *magic target names*. They're case-sensitive. If you enter a magic target name in anything other than lowercase, the link will attempt to display the document in a window with that name, creating a new window if necessary. Table 17.2 lists the magic `target` names and describes their use.

TABLE 17.2 Magic `target` Names

`target` **Name**	**Description**
`target="_blank"`	Forces the document referenced by the `<a>` tag to be loaded into a new unnamed window.
`target="_self"`	Causes the document referenced by the `<a>` tag to be loaded into the window or frame that held the `<a>` tag. This can be useful if the `<base>` tag sets the target to another frame but a specific link needs to load in the current frame.
`target="_parent"`	Forces the link to load into the `<frameset>` parent of the current document. If the current document has no parent, however, `target="_self"` will be used.
`target="_top"`	Forces the link to load into the full web browser window, replacing the current `<frameset>` entirely. If the current document is already at the top, however, `target="_self"` will be used. More often than not, when you create links to other sites on the Web, you don't want them to open within your frameset. Adding `target="_top"` to the link will prevent this from occurring.

Inline Frames

The main advantage of inline frames is that is that you can position them anywhere on a web page, just as you can other elements like images or Flash movies. You can incorporate content from another page or even another site into a page in a seamless way through the use of inline frames. Inline frames, which are specified using the `<iframe>` tag, are commonly the means by which "widgets" offered by popular websites are incorporated into other websites. For example, sites like Twitter and Facebook offer widgets that you can incorporate into your own site that are implemented using inline frames.

Here's a brief run-through of how to create floating frames. First, you define them using the `<iframe>` tag. Like images, these frames appear inline in the middle of the body of an HTML document (hence the `i` for inline). The `<iframe>` tag supports attributes like `src`, which contains the URL of the document to be displayed in the frame, and `height` and `width`, which control the size of the frame.

Table 17.3 lists the attributes of the `<iframe>` element.

TABLE 17.3 Key Attributes

Attribute	Description
`width`	Specifies the width, in pixels, of the floating frame that will hold the HTML document.
`height`	Specifies the height, in pixels, of the floating frame that will hold the HTML document.
`src`	Specifies the URL of the HTML document to be displayed in the frame.
`name`	Specifies the name of the frame for the purpose of linking and targeting.
`frameborder`	Indicates whether the frame should display a border. A value of `1` indicates the presence of a border, and a value of `0` indicates no border should be displayed.
`marginwidth`	Specifies the margin size for the left and right sides of the frame in pixels.
`marginheight`	Specifies the size of the top and bottom margins in pixels.
`noresize`	Indicates that the frame should not be resizable by the user (Internet Explorer extension).
`scrolling`	As with the `<frame>` tag, indicates whether the inline frame should include scrollbars. (This attribute can take the values `yes`, `no`, or `auto`; the default is `auto`.)
`vspace`	Specifies the height of the margin (Internet Explorer extension).
`hspace`	Specifies the width of the margin (Internet Explorer extension).

TABLE 17.3 Continued

Attribute	Description
`longdesc`	The URL for a page containing a long description of the contents of the `iframe.align`. As with the `<img>` tag, specifies the positioning of the frame with respect to the text line in which it occurs. Possible values include `left`, `middle`, `right`, `top`, and `bottom`, which is the default value. `absbottom`, `absmiddle`, `baseline`, and `texttop` are available as Internet Explorer extensions. Deprecated in favor of CSS.

Because you know how to use both regular frames and inline images, using the `<iframe>` tag is fairly easy. The following code displays one way to use the Away from My Desk pages with a floating frame. In this example, you begin by creating a page with a red background. The links that the user clicks appear on a single line, centered above the iframe. For clarity, I've placed each of the links on a separate line of code.

Following the links (which target the floating frame named `"reason"`), the code for the floating frame appears within a centered `<div>` element. As you can see in the HTML below, the floating frame will be centered on the page and will measure 450 pixels wide by 105 pixels high:

Input ▼

```
<!DOCTYPE html>
<html>
  <head>
    <title>I'm Away From My Desk</title>
    <style type="text/css" media="screen">
      body { background-color: #ffcc99; }
    </style>
  </head>
  <body>
    <h1>I'm away from my desk because ...</h1>
    <p style="text-align: center">
      <a href="reason1.html" target="reason">Reason 1</a> ¦
      <a href="reason2.html" target="reason">Reason 2</a> ¦
      <a href="reason3.html" target="reason">Reason 3</a> ¦
      <a href="reason4.html" target="reason">Reason 4</a> ¦
      <a href="reason5.html" target="reason">Reason 5</a> ¦
      <a href="reason6.html" target="reason">Reason 6</a>
    </p>

    <div style="margin: 0 auto; width: 450px;">
      <iframe name="reason" src="reason1.html" style="width: 450px; height:
105px"></iframe>
    </div>
  </body>
</html>
```

Figure 17.17 shows the result.

Output ▶

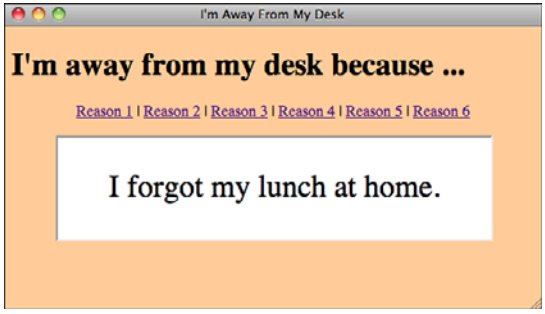


FIGURE 17.17
An inline (or floating) frame.

Opening Linked Windows with JavaScript

Pop-up windows are used all over the Web. They are often used to display advertisements, but they can be used for all sorts of other things, as well, such as creating a separate window to show help text in an application or to display a larger version of a graph that's embedded in a document. You've seen how you can use the `target` attribute to open a link in a new window, but that approach isn't flexible. You can't control the size of the window being displayed, nor which browser interface controls are displayed.

Fortunately, with JavaScript you can take more control of the process of creating new windows. You've already learned that one of the objects supported by JavaScript is `window`. It refers to the window that's executing the script. To open a new window, you use the `open` method of the `window` object. Here's a JavaScript function that opens a window:

```
function popup(url) {
  mywindow = window.open(url, 'name', 'height=200,width=400');
  return false;
}
```

The function accepts the URL for the document to be displayed in the new window as an argument. It creates a new window using the `window.open` function, and assigns that new window to a variable named `mywindow`. (I explain why we assign the new window to a variable in a bit.)

The three arguments to the function are the URL to be displayed in the window, the name for the window, and a list of settings for the window. In this case, I indicate that I want the window to be 400 pixels wide and 200 pixels tall. The name is important because if other links target a window with the same name, either via the `window.open()` function or the `target` attribute, they'll appear in that window.

At the end of the function, I return `false`. That's necessary so that the event handler used to call the function is stopped. To illustrate what I mean, it's necessary to explain how this function is called. Instead of using the `target` attribute in the `<a>` tag, the `onclick` handler is used, as follows:

```
<a href="whatever.html" target="_blank" onclick="popup('whatever.html')">Pop
up</a>
```

Ordinarily, when a user clicks the link, the browser calls the function and then goes right back to whatever it was doing before, navigating to the document specified in the `href` attribute. Returning `false` in the `popup()` function tells the browser not to continue what it was doing, so the new window is opened by the function, and the browser doesn't follow the link. If a user who had JavaScript turned off visited the page, the link to `whatever.html` would still open in a new window because I included the target attribute, too.

In the preceding example, I specified the `height` and `width` settings for the new window. There are several more options available as well, which are listed in Table 17.4.

TABLE 17.4 Settings for Pop-Up Windows

Setting	Purpose
`height`	Height of the window in pixels.
`width`	Width of the window in pixels.
`resizable`	Enable window resizing.
`scrollbars`	Display scrollbars.
`status`	Display the browser status bar.
`toolbar`	Display the browser toolbar.
`location`	Display the browser's location bar.
`menubar`	Display the browser's menu bar (Not applicable on Mac OS X.).
`left`	Left coordinate of the new window onscreen (in pixels). By default, pop-up windows are placed slightly to the right of the spawning window.
`top`	Top coordinate of the new window onscreen (in pixels). By default, pop-up windows are placed slightly below the top of the spawning window.

When you specify the settings for a window, you must include them in a comma-separated list, with no spaces anywhere. For the settings that allow you to enable or disable a browser interface component, the valid values are `on` or `off`. Here's a valid list of settings:

```
status=off,toolbar=off,location=off,left=200,top=100,width=300,height=300
```

Here's an invalid list of settings:

```
status=off, toolbar=off, location=false, top=100
```

Including spaces (or carriage returns) anywhere in your list will cause problems. It's also worth noting that when you provide settings for a new window, the browser automatically assumes a default of `off` for any on/off settings that you don't include. So, you can leave out anything you want to turn off.

Here's a complete example that uses JavaScript to create a new window:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Popup example</title>
    <script type="text/javascript">
      function popup(url) {
        var mywindow = window.open(url, 'name', 'height=200,width=400');
        return false;
      }
    </script>
  </head>
  <body>
    <h1>Popup Example</h1>
    <p>
      <a href="popup.html" onclick="popup(this.href)">Launch popup</a>
    </p>
  </body>
</html>
```


When a user clicks the Launch popup link, a new 200x400 pixel window appears with the contents of `popup.html`.

The unobtrusive approach is to skip the `onclick` attribute entirely and bind the `popup()` function to the link in your JavaScript code. First, change the link on the page to look like this:

```
<a href="popup.html" id="launchpopup">Launch popup</a>
```

Then you should edit the `<script>` tag so that it looks like this:

```
<script type="text/javascript">
      function popup(url) {
          var mywindow = window.open(url, 'name', 'height=200,width=400');
          return false;
      }
      window.onload = function () {
         var link = document.getElementById("launchpopup");
         link.onclick = function () {
             return popup(this.href);
```

```
        }
      }
</script>
```

In this case, when the page loads, I retrieve the link by its ID and then bind a new anonymous function to it that calls the original `popup()` function. Instead of hard coding the URL, I pass `this.href` to the `popup()` function so that it opens the URL in the link.

Using a library like jQuery can make things even easier. Suppose you want any link tag with the class `popup` to open a new window with the URL associated with the link. Here's the code:

```
<script type="text/javascript">
$(document).ready(function () {
    $("a.popup").click(function (event) {
        var mywindow = window.open(this.href, 'newwindow',
'height=200,width=400');
        event.preventDefault();
    });
});
</script>
```

When the page is ready, I apply the same `onclick` handler to all the links on the page with the class `popup`. The anonymous event handler opens a new window with the URL stored in `this.href`, which returns the URL in the link that the user clicked on. It then calls the `preventDefault()` method on the link, which is a sort of fancy way to return false provided by jQuery. It's used instead of just returning false because it doesn't disrupt other event handlers that may be fired in addition to this one.

Summary

If your head is hurting after this lesson, you're probably not alone. Although the basic concepts behind the use of frames are relatively straightforward, their implementation is somewhat harder to come to grips with. As a result, the best way to learn about frames is by experimenting with them.

In this lesson, you learned how to link a document to a new or existing window. In addition, you learned how to create framesets and link them together by using the tags listed in Table 17.5.

The Downside of Frames

In this lesson, I mentioned a few drawbacks of frames here and there. I want to talk about them one more time now that the discussion of frames is coming to a close. The problem with frames is that they change the concept of a web page. Unfortunately, many mechanisms that users are familiar with rely on the original concept. So when it comes to navigation, bookmarking pages, or using the browser's Back button, frames can cause confusion. There are also issues when it comes to printing, and frames can cause problems on devices with small screens, like mobile phones.

The point here isn't to say that frames should never be used, but rather that you should think about what you're trying to accomplish and decide whether another approach wouldn't work better. Frames can combine content from multiple sites into one, but in many cases, JavaScript can achieve similar effects. My advice is to use frames sparingly.

TABLE 17.5 New Tags Discussed in Lesson 17

Tag	Attribute	Description
`<base target="window">`		Sets the global link window for a document.
`<frameset>`		Defines the basic structure of a frameset.
	`cols`	Defines the number of frame columns and their width in a frameset.
	`rows`	Defines the number of frame rows and their height in a frameset.
	`<frame>`	Defines the contents of a frame within a frameset.
	`src`	Indicates the URL of the document to be displayed inside the frame.
	`marginwidth`	Indicates the size in pixels of the margin on each side of a frame.
	`marginheight`	Indicates the size in pixels of the margin above and below the contents of a frame.
	`scrolling`	Enables or disables the display of scrollbars for a frame. Values are `yes`, `no`, and `auto`.
	`noresize`	Prevents the users from resizing frames.

TABLE 17.5 Continued

Tag	Attribute	Description
	`frameborder`	Indicates whether the frameset displays borders between frames.
	`longdesc`	Specifies a URL that provides a longer description of the contents of the frameset. Used with nonvisual browsers.
	`name`	Assigns a name to the frame for targeting purposes.
`<iframe>`		Defines an inline or floating frame.
	`src`	Indicates the URL of the document to be displayed in the frame.
	`name`	Indicates the name of the frame for the purpose of linking and targeting.
	`width`	Indicates the width of the frame in pixels.
	`height`	Indicates the height of the frame in pixels.
	`marginwidth`	Indicates the width of the margin in pixels.
	`marginheight`	Indicates the height of the margin in pixels.
	`scrolling`	Enables or disables the display of scrollbars in the frame. Values are `yes`, `no`, and `auto`.
	`frameborder`	Enables or disables the display of a border around the frame. Values are `1` and `0`.
	`align`	Specifies the alignment of the frame relative to the current line of text. Values are `left`, `right`, `middle`, `top`, and `bottom` (also `absbottom`, `absmiddle`, `texttop`, and `baseline` in Internet Explorer).
`<noframes>`		Defines text to be displayed by web browsers that don't support the use of frames.

Workshop

As if you haven't had enough already, here's a refresher course of questions, quizzes, and exercises that will help you remember some of the most important points you learned in this lesson.

Q&A

Q Is there any limit to how many levels of `<frameset>` tags I can nest within a single screen?

A No, there isn't a limit. Practically speaking, however, the available window space starts to become too small to be usable when you get below about four levels.

Q What would happen if I included a reference to a frameset document within a `<frame>` tag?

A Most browsers that support frames handle such references correctly by treating the nested frameset document as a nested `<frameset>`. In fact, this technique is used regularly to reduce the complexity of nested frames.

One limitation does exist, however: You cannot include a reference to the current frameset document in one of its own frames. This situation, called *recursion*, causes an infinite loop.

Quiz

1. What are the differences between a *frameset document*, a *frameset*, a *frame*, and a *page*?
2. When you create links to pages that are supposed to load into a frameset, what attribute makes the pages appear in the right frame? (Hint: It applies to the `<a>` element.)
3. When a web page includes the `<frameset>` element, what element cannot be used at the beginning of the HTML document?
4. What two attributes of the `<frameset>` tag divide the browser window into multiple sections?
5. What attribute of the `<frame>` tag defines the HTML document that first loads into a frameset?

Quiz Answers

1. A *frameset document* is the HTML document that contains the definition of the frameset. A *frameset* is the portion of the frameset document that is defined by the `<frameset>` tag, which instructs the browser to divide the window into multiple sections. A *frame* is one of the sections, or windows, within a frameset. The *page* is the web document that loads within a frame.
2. The `target` attribute of the `<a>` tag directs linked pages to load into the appropriate frame.
3. When a web page includes the `<frameset>` element, it cannot include the `<body>` element at the beginning of the page. They're mutually exclusive.
4. The `cols` and `rows` attributes of the `<frameset>` tag divide the browser window into multiple frames.
5. The `src` attribute of the `<frame>` tag defines the HTML document that first loads into the frameset.

Exercises

1. Create a frameset that divides the browser window into three sections, as follows:
   - The left section of the frameset will be a column that spans the entire height of the browser window and will take up one-third of the width of the browser window. Name this frame `contents`.
   - Divide the right section of the frameset into two rows, each taking half the height of the browser window. Name the top section `top` and the bottom section `bottom`.

2. For the preceding frameset, create a page that you will use for a table of contents in the left frame. Create two links on this page, one that loads a page in the top frame and another that loads a page in the bottom frame.

LESSON 18
Writing Good Web Pages: Do's and Don'ts

You won't learn about any HTML tags in this lesson, or how to convert files from one strange file format to another. You're mostly done with the HTML part of web page design. Next come the intangibles. These are the things that separate your pages from those of a designer who just knows the tags and flings text and graphics around and calls it a site.

Armed with the information from the lessons so far, you could put this book down now and go off and merrily create web pages to your heart's content. However, if you stick around, you will be able to create even *better* web pages. Do you need any more incentive to continue reading?

This lesson includes hints for creating well-written and well-designed web pages, and it highlights do's and don'ts concerning the following:

- How to make decisions about compliance with web standards
- How to write your web pages so that they can be easily scanned and read
- Issues concerning the design and layout of your web pages
- When and why you should create links
- How to use images effectively
- Other miscellaneous tidbits and hints

Standards Compliance and Web Browsers

In some ways, we're through the dark ages when it comes to cross-browser issues. The generation of browsers that includes Netscape Navigator 4 and Internet Explorer 4 probably marked the low point for web developers in terms of deciding how pages should be written. In the heat of the browser wars, Netscape and Microsoft were adding new features to their browsers hand over fist, with no regard for published standards. These features tended to be at odds with each other, and for web designers to create complex pages that worked in both popular browsers, they had to use some really awful techniques to make things look okay. Worse, the differences made things like *Cascading Style Sheets (CSS)* and JavaScript very difficult to use. Even if you could get things to work properly, the work involved was immense.

Currently, browser makers are working together to a greater extent than ever before. The current versions of Internet Explorer, Firefox, Safari, Chrome, and Opera all offer strong standards support, and even the browsers for popular mobile phone platforms like Android and iPhone have standards-based browsers. Given the strong standards support in current browsers, the biggest question most developers face is how they want to deal with Internet Explorer 6. Version 6 of Internet Explorer was released in 2001, and it still has a number of users, although that number continues to fall. The main problem with that browser is that it deals with CSS differently than later versions of Internet Explorer as well as other browsers, and those differences can make life painful for web designers. Many websites are dropping support for Internet Explorer 6 entirely because it differs so greatly from standards-based browsers.

Despite the fact that the browser picture is relatively clear these days, there are still cases where browsers differ in terms of capabilities. The most obvious example is mobile browsers. They have smaller screens than regular computers, and even the most advanced mobile browsers have fewer capabilities than desktop browsers. There are also millions of people with mobile devices that can access the Web but that have very limited support for CSS and JavaScript. Even their support for *Hypertext Markup Language (HTML)* markup is somewhat limited.

Progressive Enhancement

Progressive enhancement is a popular approach to creating web pages. It describes an approach that enables web designers to use the latest and greatest technology available without leaving people using browsers with limited capabilities behind. The idea is that

you start with simple but completely functional web pages and then layer on enhancements that add to the experience for people with newer or more capable browsers without breaking the experience for the people using their mobile device or the old browser that their employer forces them to use.

You'll want to start with valid, standards-compliant HTML when you're creating web pages. There are a number of relevant standards, including HTML5, which is not yet complete. I talk about the differences among them shortly. Regardless, you'll want to choose one of them and make sure all of your pages adhere to it.

Your initial pages should consist only of HTML markup with no JavaScript or CSS, and they should look fine and work properly. All of your navigation should be present and should work properly. Your main page content should be visible. In other words, you should start out with a fully functional, very plain website. This ensures that your site will work for even the most rudimentary mobile browsers and also for vision-challenged users with screen readers. Taking this approach also ensures that your markup reflects the content of your site rather than how you want to present it.

When that is complete, you can start layering on the more advanced functionality. First, implement the visual design for your website using CSS. You can organize your content into columns, transform your navigation so that it is presented in collapsible menus, and add visual interest by using image backgrounds for various elements. Given the robust support for CSS in the current browsers, there should be no need to use HTML to define the appearance of your website. You might find you need to add container elements to your page that provide the necessary structure for your styles. For example, if your page layout is split into columns, it will be necessary to add `<div>` tags for the contents of each column. Fortunately, such tags do not create any visual differences unless they are styled, so your page's appearance will not be altered for users who don't have CSS support.

Finally, add dynamic technology like JavaScript or Flash. In Lesson 15, "Using JavaScript in Your Pages," I discussed unobtrusive JavaScript. That approach complements progressive enhancement. When you add JavaScript to the page, make sure the page provides some minimum level of functionality without the JavaScript, and then use JavaScript to enhance that baseline functionality. For example, if your page includes collapsed elements that can be expanded with JavaScript, make sure to start out with them expanded on the page, and then collapse them when the page loads using JavaScript. That way, content will not be permanently hidden from users who don't have JavaScript.

There are also methods for adding Flash to pages in unobtrusive ways. Many websites present all their content using Flash, but it is possible to do so while also providing

HTML as an alternative, by starting with HTML content on the page and using JavaScript to hide that content and present the Flash movie instead if the user has a browser that supports Flash. For example, many mobile devices do not support Flash. If you are building a website for a restaurant in Flash, it makes sense to present essential information like the restaurant's hours, location, and phone number in HTML, and then optionally hide that information and replace it with your Flash movie if the Flash player is available. If it isn't, users can still access the important information on the site.

This is what progressive enhancement is all about. It ensures that everyone with a browser of any kind can view your site, while the site still provides an enhanced experience for those who can benefit from it.

Choosing a Document Type

In light of these different approaches to writing HTML, let's look at the current HTML standards. The HTML 4.01 and XHTML 1.0 definitions include three flavors of HTML, and the main differences between HTML 4.01 and XHTML 1.0 have been noted throughout this book. The most important of them is that XHTML documents must be valid *Extensible Markup Language (XML)*, whereas that is not required for HTML 4.01. The three flavors are as follows:

- **HTML 4.01 or XHTML 1.0 Transitional** is geared toward the web developer who still uses deprecated tags rather than migrating to CSS. For example, if you want to write valid pages that include `<font>` tags, and attributes of the `<body>` tag like `bgcolor` or `text`, this is your best bet.
- **HTML 4.01 or XHTML 1.0 Frameset** is a strict specification that includes the frame-related tags. You should use this standard when you're writing framed pages.
- **HTML 4.01 or XHTML 1.0 Strict** is for people who don't have deprecated tags in their documents. This specification basically mandates that tags are used strictly for page structure, and that all your look-and-feel modifications are accomplished using CSS.

Most web developers stick with the XHTML standard that is most applicable to them, with a strong preference toward XHTML 1.0 Strict. At one time, significant differences existed between browsers in terms of the tags they supported. These days, all the browsers support an almost identical set of tags. Instead, the differences lie mainly in how they deal with CSS. Regardless of the strategy you use for creating content, the key to making sure everything works is to test it as widely as possible. If you're concerned about how your page looks in Internet Explorer 6, test it in Internet Explorer 6. After a

while, you'll get the feel for how things are going to look, but at first you'll need to test exhaustively. Even after you've been at it for a while, more testing never hurts. You might have a favorite browser, but as a web designer, you can't afford to ignore the others that aren't your favorites.

HTML5

The HTML5 standard is not finished, but it's close enough to being finished that browser makers are already starting to support some HTML5 features. The question when you're creating web pages is whether you should stick with XHTML 1.0 or move on to HTML5. In this book, I have used the new HTML5 document type for all the pages that don't include elements that aren't supported in HTML5. It works fine with all the current browsers and makes sure your pages are ready for HTML5 when it arrives. I've also avoided elements and attributes that have been dropped from HTML5.

One thing that's different about HTML5 from XHTML 1.0 is that HTML5 documents are not required to be valid XML. Valid XML is supported if you choose to use it, but you can also use the older, HTML 4.01 style if you prefer.

In the next lesson, I take a deeper look at what HTML5 offers and the differences between it and XHTML 1.0.

Validating Your Pages

It's all well and good to attempt to write valid pages, but how do you know whether you've succeeded? It's easy enough to determine whether your pages look okay in your browser, but verifying that they're valid is another matter. Fortunately, the W3C, which is responsible for managing the HTML recommendations, also provides a service to validate your pages. It's a web application that enables you to upload an HTML file or to validate a specific URL to any W3C recommendation. The URL is http://validator.w3.org/.

Figure 18.1 is a screenshot of the validator in action. I've sent it off to validate http://www.informit.com/.

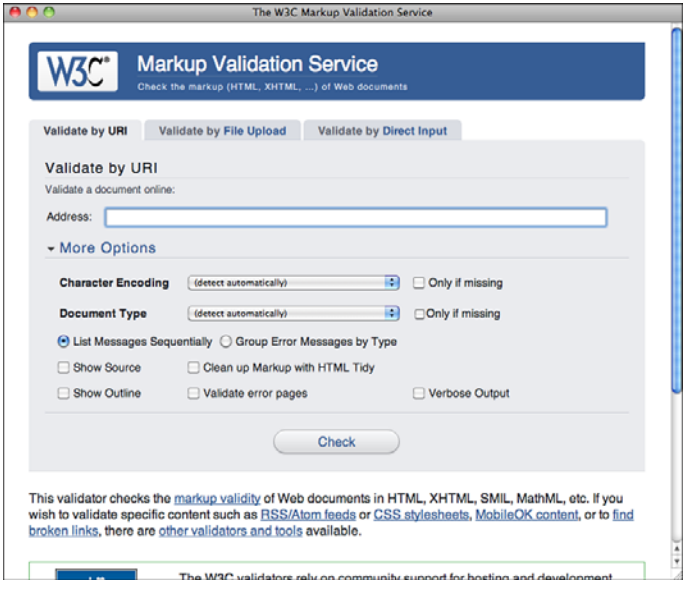


FIGURE 18.1
The W3 Validator.

If you look closely at the screenshot, you can see that at the time I validated this page, it came out with 42 errors, all of which are violations of the XHTML 1.0 Transitional recommendation. The page looks fine in the popular browsers, but it's not in sync with the recommendation. The errors generally fall into two categories: missing closing tags (usually for the `<img>` tag) and missing required attributes (mainly the `alt` attribute for `<img>` tags). This page also includes some invalid attributes for tags. One common error that appears on this page refers to an entity with no system identifier, and a longer set of errors related to the same bit of markup. The problem here is that the validator expects the character `&` to be the beginning of an entity, but on this page, it is part of the URL for a link. To avoid these errors, the ampersands in links need to be encoded as `&`.

You can see this entity-related error in Figure 18.2.

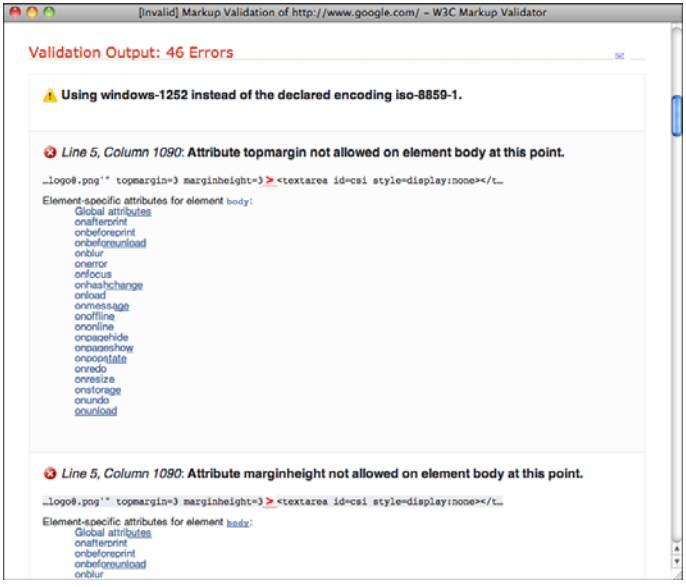


FIGURE 18.2
Error messages produced by the W3 Validator.

In the screenshot, you'll see that each error is accompanied by the subset of the page's source where the error appears along with an explanation for the error. After you've fixed all the errors on your page, you'll be privileged enough to see the message indicating that your page is valid, as shown in Figure 18.3.

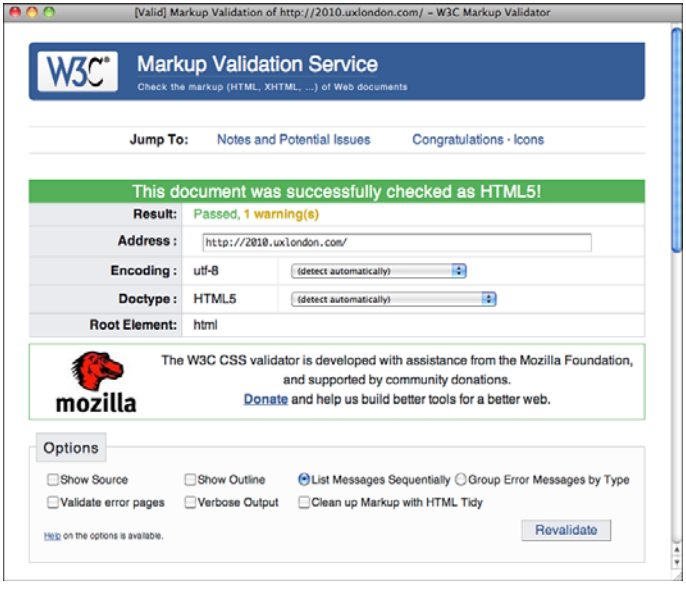


FIGURE 18.3
The W3 Validator acknowledges a job well done.

HTML Tidy

If you have some existing pages and you're not up for the challenge of turning them into valid HTML, there's a program that will not only validate your pages, but also correct them as best as it can. This program, HTML Tidy, was originally written by a staffer at the W3C named David Raggett and has since been handed over to the community for maintenance. You can obtain it at http://tidy.sourceforge.net/.

Tidy accepts a number of command-line options that enable you to indicate how you want your page to be validated and modified. For example, if you want to convert your pages to XHTML, you can call it like this:

```
tidy -asxhtml myfile.html
```

Tidy will strip out all the deprecated `<font>` tags in your document, along with other tags that can be replaced by CSS properties if you pass it the `-clean` option, like this:

```
tidy -clean myfile.html
```

After you've run Tidy on your files, they may still need some cleaning up, but Tidy will fix all the obvious errors.

You might also want to look for support for HTML Tidy in your text editor if it has HTML-specific features. Tidy is also available as a library that can be incorporated into other tools, and many offer the ability to clean up your files as you edit them.

Writing for Online Publication

Writing on the Web is no different from writing in the real world. Although it's not committed to hard copy, it's still published and is still a reflection of you and your work. Because your writing is online and your visitors have many other options when it comes to finding something to read, you'll have to follow the rules of good writing that much more closely.

Because of the vast quantities of information available on the Web, your visitors aren't going to have much patience if your web page is poorly organized or full of spelling errors. They're likely to give up after the first couple of sentences and move on to someone else's page. After all, there are several million pages out there. No one has time to waste on bad pages.

I don't mean that you have to go out and become a professional writer to create a good web page, but I give you a few hints for making your web page easier to read and understand.

Write Clearly and Be Brief

Unless you're writing the Great American Web Novel, your visitors aren't going to linger lovingly over your words. You should write as clearly and concisely as you possibly can, present your points, and then stop. Obscuring what you want to say with extra words just makes figuring out your point more difficult.

If you don't have a copy of Strunk and White's *The Elements of Style*, put down this book right now and go buy that book. Read it, reread it, memorize it, inhale it, sleep with it under your pillow, show it to all your friends, quote it at parties, and make it your life. You'll find no better guide to the art of good, clear writing than *The Elements of Style*.

Organize Your Pages for Quick Scanning

Even if you write the clearest, briefest, most scintillating prose ever seen on the Web, chances are good that your visitors won't start at the top of your web page and carefully read every word down to the bottom.

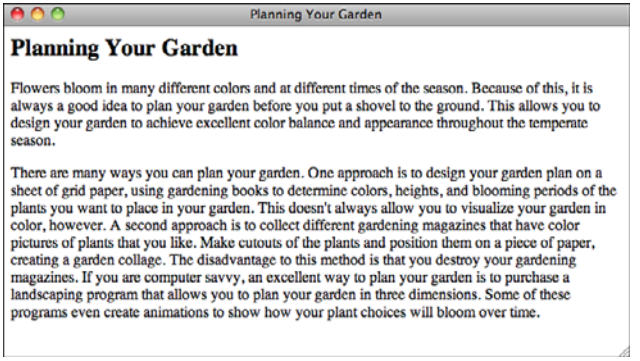
In this context, *scanning* is the first quick look your visitors give to each page to get the general gist of the content. Depending on what your users want out of your pages, they may scan the parts that jump out at them (headings, links, other emphasized words), perhaps read a few contextual paragraphs, and then move on. By writing and organizing your pages for easy "scannability," you can help your visitors get the information they need as quickly as possible.

To improve the scannability of your web pages, follow these guidelines:

- **Use headings to summarize topics**—Note that this book has headings and subheadings. You can flip through quickly and find the parts that interest you. The same concept applies to web pages.
- **Use lists**—Lists are wonderful for summarizing related items. Every time you find yourself saying something like "each widget has four elements" or "use the following steps to do this," the content after that phrase should be in an ordered or unordered list.
- **Don't forget link menus**—As a type of list, the link menu has all the same advantages of lists for scannability, and it doubles as an excellent navigation tool.
- **Don't bury important information in text**—If you have a point to make, make it close to the top of the page or at the beginning of a paragraph. Forcing readers to sift through a lot of information before they get to what's important means that many of them won't see the important stuff at all.
- **Write short, clear paragraphs**—Long paragraphs are harder to read and make gleaning the information more difficult. The further into the paragraph you put your point, the less likely it is that anybody will read it.

Figure 18.4 shows the sort of writing technique that you should avoid.

FIGURE 18.4
DON'T: A web page that is difficult to scan.



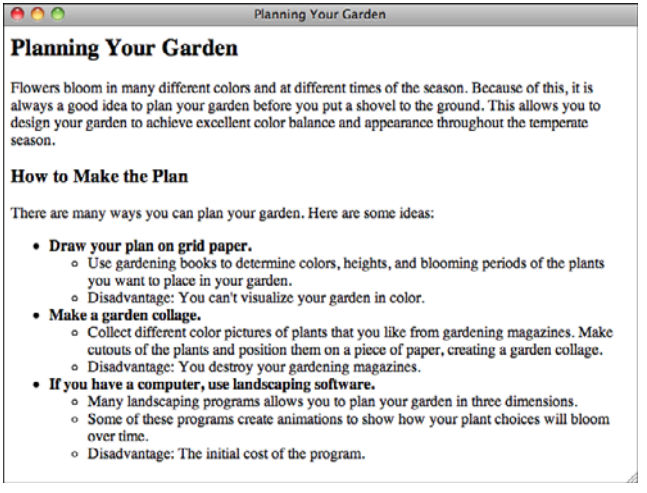
Because all the information on this page is in paragraph form, your visitors have to read both paragraphs to find out what they want and where they want to go next.

How would you improve the example shown in Figure 18.4? Try rewriting this section so that visitors can better find the main points from the text. Consider the following:

- These two paragraphs actually contain three discrete topics.
- The ways to plan the garden would make an excellent nested list.

Figure 18.5 shows what an improvement might look like.

FIGURE 18.5
DO: An improvement to the difficult-to-scan web page.



Make Each Page Stand on Its Own

As you write, keep in mind that your visitors could jump to any of your web pages from anywhere. For example, you can structure a page so that section four distinctly follows section three and has no other links to it. Then someone you don't even know might create a link to the page starting at section four. From then on, visitors could find themselves at section four without even knowing that section three exists.

Be careful to write each page so that it stands on its own. The following guidelines will help:

- **Use descriptive titles**—The title should provide not only the direct subject of this page, but also its relationship to the rest of the pages on the site.
- **Provide a navigational link**—If a page depends on the one before it, provide a navigational link back to that page (and also a link up to the top level, preferably).
- **Avoid initial sentences such as the following**—"You can get around these problems by…," "After you're done with that, do this…," and "The advantages to this method are…." The information referred to by *these*, *that*, and *this* are off on some other page. If these sentences are the first words your visitors see, they're going to be confused.

Be Careful with Emphasis

Use emphasis sparingly in your text. Paragraphs with a whole lot of words in **boldface** or *italics* or ALL CAPS are hard to read, whether you use them several times in a paragraph or to emphasize long strings of text. The best emphasis is used only with small words such as *and*, *this*, or *but*.

Link text also is a form of emphasis. Use single words or short phrases for link text. Do not use entire passages or paragraphs. Figure 18.6 illustrates a particularly bad example of too much emphasis obscuring the rest of the text.

By removing some of the boldface and using less text for your links, you can considerably reduce the amount of clutter in the paragraph, as you can see in Figure 18.7.

Be especially careful of emphasis that moves or changes, such as marquees, blinking text, or animation. Unless the animation is the primary focus of the page, use movement and sound sparingly.

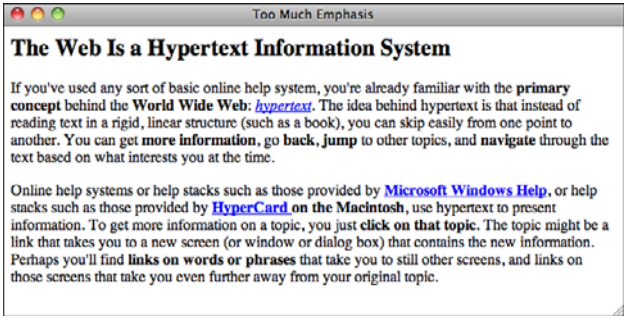


FIGURE 18.6
DON'T: Too much emphasis.

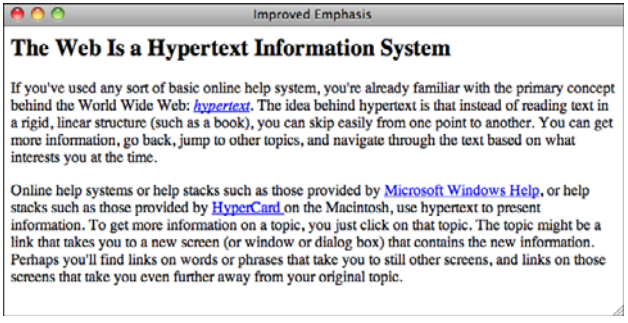


FIGURE 18.7
DO: Less emphasis.

Don't Use Browser-Specific Terminology

Avoid references in your text to specific features of specific browsers. For example, don't use the following wording:

- **"Click here"**—What if your visitors are surfing without a mouse? Users of mobile phones with touch screens "tap" on links. A more generic phrase is "Follow this link." (Of course, you should avoid the "here" syndrome in the first place, which neatly gets around this problem, too.)
- **"To save this page, pull down the File menu and select Save"**—Each browser has a different set of menus and different ways of accomplishing the same actions. If at all possible, do not refer to specifics of browser operation in your web pages.
- **"Use the Back button to return to the previous page"**—Each browser has a different set of buttons and different methods for going back. If you want your visitors to go back to a previous page or to any specific page, link those pages.

Spell Check and Proofread Your Pages

Spell checking and proofreading may seem like obvious suggestions, but they bear mentioning given the number of pages I've seen on the Web that obviously haven't had either.

The process of designing a set of web pages and making them available on the Web is like publishing a book, producing a magazine, or releasing a product. Publishing web pages is considerably easier than publishing books, magazines, or other products, of course, but just because the task is easy doesn't mean your product should be sloppy.

Thousands of people may be reading and exploring the content you provide. Spelling errors and bad grammar reflect badly on you, on your work, and on the content you're describing. It may be irritating enough that your visitors won't bother to delve any deeper than your home page, even if the subject you're writing about is fascinating.

Proofread and spell check each of your web pages. If possible, have someone else read them. Often other people can pick up errors that you, the writer, can't see. Even a simple edit can greatly improve many pages and make them easier to read and navigate.

Design and Page Layout

With the introduction of technologies such as style sheets and *Dynamic HTML (DHTML)*, people without a sense of design have even more opportunities to create a site that looks simply awful.

Probably the best rule of web design to follow at all times is this: *Keep the design of each page as simple as possible*. Reduce the number of elements (images, headings, and rule lines) and make sure that visitors' eyes are drawn to the most important parts of the page first.

Keep this cardinal rule in mind as you read the next sections, which offer some other suggestions for basic design and layout of web pages.

Use Headings as Headings

Headings tend to be rendered in larger or bolder fonts in graphical browsers. Therefore, using a heading tag to provide some sort of warning, note, or emphasis in regular text can be tempting (see Figure 18.8).

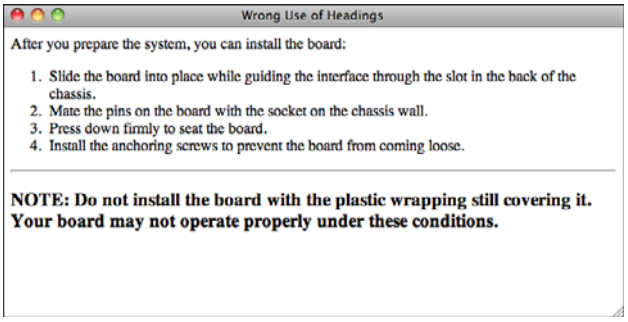


FIGURE 18.8
DON'T: The wrong way to use headings.

Headings stand out from the text and signal the start of new topics, so they should be used only as headings. If you want to emphasize a particular section of text, consider using a small image, a rule line, or some other method of emphasis instead. Remember that you can use CSS to change the color, background color, font size, font face, and border for a block of text. Figure 18.9 shows an example of the text from Figure 18.8 with a different kind of visual emphasis.

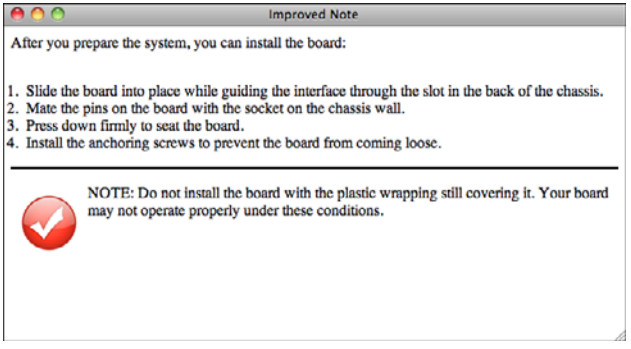


FIGURE 18.9
DO: The right way to use headings.

Group Related Information Visually

Grouping related information within a page is a task for both writing and design. As I suggested in the "Writing for Online Publication" section, grouping related information under headings improves the scannability of that information. Visually separating each section from the others helps to make it distinct and emphasizes the relatedness of the information.

If a web page contains several sections, find a way to separate those sections visually—for example, with a heading, a rule line, or tables, as shown in Figure 18.10.

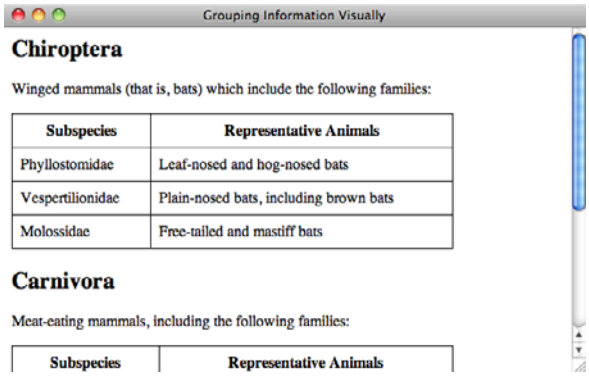


FIGURE 18.10
Do: Separate sections visually.

Use a Consistent Layout

When you're reading a book, each page or section usually has the same layout. The page numbers are placed where you expect them, and the first word on each page starts in the same place.

The same sort of consistent layout works equally well on web pages. Having a single look and feel for each page on your website is comforting to your visitors. After two or three pages, they'll know what the elements of each page are and where to find them. If you create a consistent design, your visitors can find the information they need and navigate through your pages without having to stop at every page and try to find where certain elements are located.

Consistent layout can include the following:

- **Consistent page elements**—If you use second-level headings (`<h2>`) on one page to indicate major topics, use second-level headings for major topics on all your pages. If you have a heading and a rule line at the top of your page, use that same layout on all your pages.
- **Consistent forms of navigation**—Put your navigation menus in the same place on every page (usually the top or the bottom of the page, or even both), and use the same number of them. If you're going to use navigation icons, make sure that you use the same icons in the same order for every page.
- **The use of external style sheets**—You should create a master style sheet that defines background properties, text and link colors, font selections and sizes, margins, and more. The appearance of your pages maintains consistency throughout your site.

Using Links

Without links, web pages would be dull and finding anything interesting on the Web would be close to impossible. In many ways, the quality of your links can be as important as the writing and design of your actual pages. Here's some friendly advice on creating and using links.

Use Link Menus with Descriptive Text

As I've noted throughout this book, using link menus is a great way of organizing your content and the links on a page. If you organize your links into lists or other menu-like structures, your visitors can scan their options for the page quickly and easily.

Just organizing your links into menus might not be enough, however. Make sure that your descriptions aren't too short. For example, using menus of filenames or other marginally descriptive links in menus can be tempting (see Figure 18.11).

FIGURE 18.11
DON'T: A poor link menu.

Each link describes the actual page to which it points, but it doesn't describe the *content* of the page. How do visitors know what's on the other side of the link, and how can they decide whether they're interested in it from the limited information you've given them? Of these three links, only the last (`pesto-recipe.txt`) gives the visitors a hint about what they'll see when they jump to that file.

A better plan is either to provide some extra text describing the content of the file, as shown in Figure 18.12, or to avoid the filenames altogether. Just describe the contents of the files in the menu with the appropriate text highlighted, as shown in Figure 18.13.

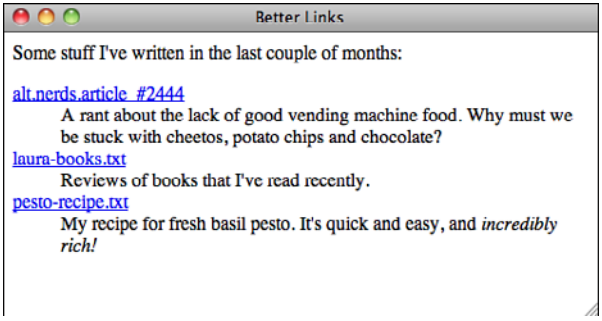


FIGURE 18.12
DO: A better link menu.

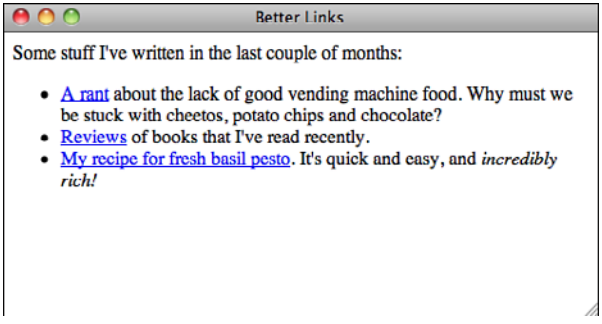


FIGURE 18.13
DO: Another better link menu.

Either one of these forms is better than the first. They both give your visitors more clues about what's on the other side of the link.

Use Links in Text

The best way to provide links in text is to first write the text as if it isn't going to have links at all—for example, as if you were writing it for hard copy. Then you can highlight the appropriate words that will link to other pages. Make sure that you don't interrupt the flow of the page when you include a link. The text should stand on its own. That way, the links provide additional or tangential information that your visitors can choose to follow or ignore at their own whim.

Figure 18.14 shows another example of using links in text. Here the text isn't particularly relevant; it's just there to support the links. If you're using text just to describe links, consider using a link menu instead of a paragraph. Instead of having to read the entire paragraph, your visitors can skim for the links that interest them.

FIGURE 18.14
DON'T: Links in text that don't work well.

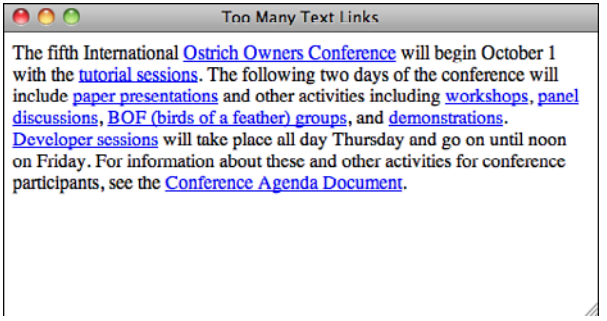


Figure 18.15 shows one way to restructure the previous example. The most important items on the page are the name of the conference, the events, and the dates on which they occur. You can restructure the page so that this information stands out. As you can see in Figure 18.15, presenting the events in a preformatted text table makes the important information stand out from the rest.

FIGURE 18.15
DO: Restructuring the links in the text.

Probably the easiest way to figure out whether you're creating links within text properly is to print out the formatted web page from your browser. In hard copy, without hypertext, does the paragraph still make sense? If the page reads funny on paper, it'll read funny online, too.

The revisions don't always have to be as different as they are in this example. Sometimes, a simple rephrasing of sentences can make the text on your pages more readable and more usable, both online and when printed.

Avoid the "Here" Syndrome

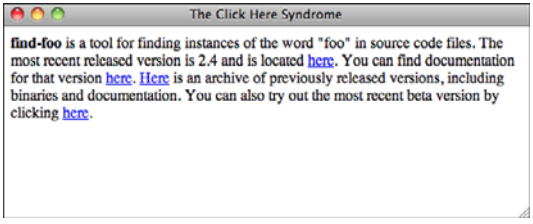
A common mistake that many web authors make when creating links in body text is using the "here" syndrome. This is the tendency to create links with a single highlighted word (`here`) and to describe the links somewhere else in the text. Look at the following examples, with underlining to indicate link text:

> Information about ostrich socialization is contained here.
> Follow this link for a tutorial on the internal combustion engine.

Because links are highlighted on the web page, the links visually pop out more than the surrounding text (or *draw the eye*, in graphic design lingo). Your visitors will see the link first, before reading the text. Try creating links this way.

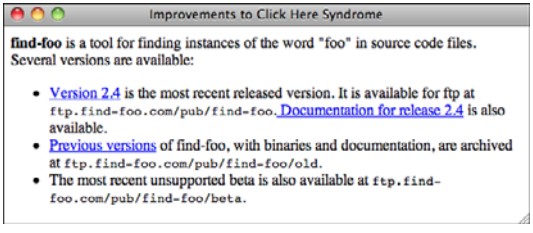
Figure 18.16 shows a particularly heinous example of the "here" syndrome. Close your eyes, open them quickly, pick a *here* in the figure at random, and then see how long it takes you to find out what the *here* is for.

FIGURE 18.16
DON'T: The "here" syndrome.



Now try the same exercise with a well-organized link menu of the same information, as shown in Figure 18.17.

FIGURE 18.17
DO: The same page reorganized.



Because "here" says nothing about what the link is used for, your poor visitors have to search the text before and after the link itself to find out what's supposed to be "here." In paragraphs that have many occurrences of *here* or other nondescriptive links, matching up the links with what they're supposed to link to becomes difficult. This forces your visitors to work harder to figure out what you mean.

To Link or Not to Link

Just as with graphics, every time you create a link, consider why you're linking two pages or sections. Is the link useful? Does it give your visitors more information or bring them closer to their goal? Is the link relevant in some way to the current content?

Each link should serve a purpose. Just because you mention the word *coffee* on a page about some other topic, you don't have to link that word to the coffee home page. Creating such a link may seem cute, but if a link has no relevance to the current content, it just confuses your visitors.

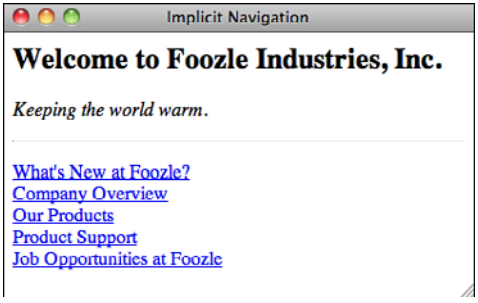
The following list describes some of the categories of useful links in web pages. If your links don't fall into one of these categories, consider the reasons why you're including them in your page:

- Explicit navigation links indicate the specific paths that visitors can take through your web pages: forward, back, up, and home. These links are often indicated by navigation icons, as shown in Figure 18.18.

FIGURE 18.18
Explicit navigation links.

- Implicit navigation links (see Figure 18.19) are different from explicit navigation links because the link text implies, but does not directly indicate, navigation between pages. Link menus are the best example of this type of link. The highlighting of the link text makes it apparent that you'll get more information on this topic by selecting the link, but the text doesn't necessarily say so. Note the major difference between explicit and implicit navigation links: If you print a page containing both, you won't pick out the implicit links.

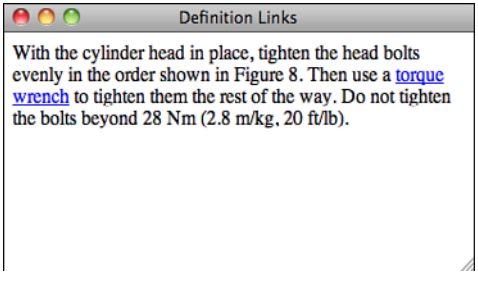
FIGURE 18.19
Implicit navigation links.



Implicit navigation links also can include tables of contents or other overviews made up entirely of links.

- Definitions of words or concepts make excellent links, particularly if you're creating large networks of pages that include glossaries. By linking the first instance of a word to its definition, you can explain the meaning of that word to visitors who don't know what it means without distracting those who do. Figure 18.20 shows an example of this type of link. (You could also use a DHTML effect to display the definition without requiring the user to follow the link.)

FIGURE 18.20
Definition links.



- Finally, links to tangents and related information are valuable when the text content will distract from the main purpose of the page. Think of tangent links as footnotes or endnotes in printed text (see Figure 18.21). They can refer to citations to other works or to additional information that's interesting but isn't necessarily directly relevant to the point you're trying to make.

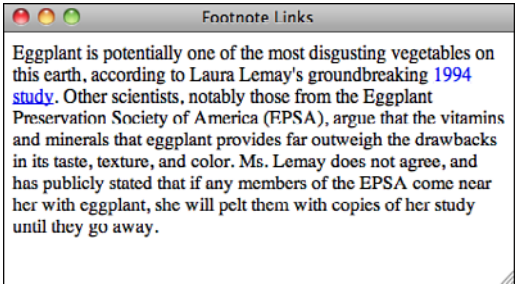


FIGURE 18.21
Footnote links.

Be careful that you don't get carried away with definitions and tangent links. You might create so many tangents that your visitors spend too much time following links elsewhere to get the point of your original text. Resist the urge to link every time you possibly can, and link only to tangents that are relevant to your own text. Also, avoid duplicating the same tangent—for example, linking every instance of the letters *WWW* on your page to the WWW Consortium's home page. If you're linking twice or more to the same location on one page, consider removing most of the extra links. Your visitors can select one of the other links if they're interested in the information.

NOTE

Thanks to Nathan Torkington for his "Taxonomy of Tags," published on the `www-talk` mailing list, which inspired this section.

Using Images

In Lesson 9, "Adding Images, Color, and Backgrounds," you learned all about creating and using images in web pages. This section summarizes many of those hints.

Don't Overuse Images

Be careful about including a large number of images on your web page. Besides that each image adds to the amount of time it takes to load the page, having too many images on the same page makes it look cluttered and distracts from the point you're trying to get across. Sometimes, though, people think that the more images they include on a page, the better it is. Figure 18.22 shows such an example.

FIGURE 18.22
DON'T: Images that are too large.

Remember the hints I gave you in Lesson 9. Consider how important each image is before you put it on the page. If an image doesn't directly contribute to the content, consider leaving it out.

Keep Images Small

Keep in mind that each image you use is a separate network connection and takes time to load over a network. This means that each image adds to the total time it takes to view the page. Try to reduce the number of images on the page, and keep them small both in file size and in actual dimensions. In particular, keep the following hints in mind:

- The entire page (text, CSS, JavaScript, and images) shouldn't take more than 10 seconds to load; otherwise, your visitors may get annoyed and move on without reading your page. Strive to achieve that size by keeping your images small and just as importan, reducing the number of images on your page. The browser has to open a new connection to your server for every image on your page, and there is overhead associated with those connections.
- For larger images, consider using thumbnails on your main page and then linking to the images rather than putting them inline.
- Save your image in both the PNG and GIF formats to see which creates a smaller file for the type of image you're using. You might also want to increase the level of compression for your JPEG images or reduce the number of colors in the palette of the GIF images to see whether you can save a significant amount of space without adversely affecting image quality.

- You can reduce the physical size of your images by cropping them (using a smaller portion of the overall image) or scaling (shrinking) them. When you scale an image, you might lose some of the detail.

CAUTION

Remember that reducing the size of your images using the `height` and `width` CSS properties or the `height` and `width` attributes of the `<img>` tag only makes them take up less space on the page; it doesn't affect the size of the image file or the download speed. It also has a tendency of making your images just look bad.

With the preceding suggestions in mind, take a second look at the images on your page. How can you put the page shown in Figure 18.22 on a diet and improve its appearance?

- The background of the page can be dropped. It makes the page hard to read, and the tiling of the image is not good. This makes the page simpler and helps it download more quickly.
- There are a couple of problems with the banner image. The first is that it's large, and the second is that it has no semantic meaning. It is purely decorative. The current style is to put things like page headings in regular tags and then use CSS backgrounds to add images where appropriate. The banner image will be replaced by a heading with a background image. This approach is friendlier to search engines and more accessible.
- Those horizontal rules are a *big* problem. First, there are too many of them. Second, they overpower the page heading because of their size. Third, they distract from the list of items because they create separation between them. I've removed them from the updated page.
- The bullets that appear before each list item are way too large. They could stand to be cut down to half their size. As a rule, most bullets are kept to 30×30 pixels or less. In the updated image, I've used smaller bubbles as bullets in the list.

All the improvements I've suggested here are shown in Figure 18.23.

FIGURE 18.23
DO: Better use of images.

Watch Out for Assumptions About Your Visitors' Hardware

Many web designers create problems for their visitors by making a couple of careless assumptions about their hardware. When you're developing web pages, be kind and remember that not everyone has the same screen and browser dimensions as you do.

Just because that huge image you created is narrow enough to fit in your browser doesn't mean that it'll fit in someone else's. An image that's too wide is annoying because the visitors need to resize their windows or scroll sideways.

Most developers limit the overall width of their pages to 750 pixels or 950 pixels, and for the sake of readability, limit the width of containers used to display text to 500 or 600 pixels. Pages meant to be displayed on mobile devices need to be even smaller.

Be Careful with Backgrounds and Link Colors

Using CSS, you can use background colors and patterns on your pages and change the color of the text. This can be tempting, but be careful. Changing the page and font colors and providing fancy backdrops can quickly and easily make your pages entirely unreadable. The following are some hints for avoiding these problems:

- **Make sure you have enough contrast between the background and foreground (text) colors**—Low contrast can be hard to read. Also, light-colored text on a dark background is harder to read than dark text on a light background.
- **Increasing the font size of all the text on your page can sometimes make it more readable on a low-contrast background**—You can use CSS `g` to increase the default font size for your page.

- **If you're using a background image, make sure that it doesn't interfere with the text**—Some images may look interesting on their own but can make text difficult to read when you put it on top of them. Keep in mind that backgrounds are supposed to be in the *background*. Subtle patterns are always better than wild patterns. Your visitors are visiting your pages for their content, not to marvel at your ability to create faux marble in your favorite image editor.

When in doubt, try asking a friend to look at your pages. Because you're already familiar with the content, you may not realize how hard your pages are to read. Someone who hasn't read them before will tell you that your text colors are too close to your background color, or that the background image is interfering with the text. Of course, you'll have to find a friend who will be honest with you.

Making the Most of CSS and JavaScript

Web design these days is about minimal markup, styled using CSS. Sticking with the following rules of thumb will make sure that your sites load quickly and efficiently.

Put Your CSS and JavaScript in External Files

Nearly all browsers maintain a cache of recently loaded content, the more content on your site that can be cached, the more quickly pages on your site after the first one will load. You should put your styles in external style sheets and your JavaScript in linked scripts whenever you can. Linked files will be cached when users visit the first page on your site and then will be retrieved from the cache on subsequent page views.

There are also advantages to this approach in terms of saving you time, too. If the styles for each page on your site live in `<style>` tags on those pages, you have to update every page when you decide to make a change. It's much easier to make those changes in an external style sheet.

You can even include styles for specific pages in a single external style sheet if you use the class and id attributes cleverly. The `<body>` tag for a page can have a class or id, just like any other element. So if you want to the pages in the news section of your site to have one background color and the pages in the "about us" section to have another, you could use this `<body>` tag for "about us":

```
<body class="aboutus">
```

For news, you'd use this one:

```
<body class="news">
```

And then in your style sheet, you include the following styles:

```
body.aboutus { background-color: black; }
body.news { background-color: grey; }
```

The same rule applies to JavaScript, too. If you use unobtrusive JavaScript, discussed in Lesson 15, "Using JavaScript in Your Pages," you can often put all the JavaScript for a site in a single file.

TIP

If possible, you will want to keep the size of your external CSS and JavaScript files under 25k. Apple's iPhone does not cache files larger than 25k, so it's better to split external files up as necessary if they are larger than that.

Location Matters

HTML 4, XHTML 1.0, and HTML5 all require links to external style sheets and the `<style>` tag to reside within the `<head>` element. You should be sure to follow this rule, because placing style sheets elsewhere in your document can cause your pages to take longer to display. By the same token, whenever possible, it's best to put `<script>` tags at the bottom of your document. When browsers are downloading an external script file, they don't try to download any other page elements. This can slow down overall page loading time. Putting the scripts last on the page enables it to download everything else on the page in parallel before it gets to the scripts. It can make your pages load a bit more quickly.

Shrink Your CSS and JavaScript

When you finish with your CSS and JavaScript, it's a good idea to compress them so that they download more quickly. Yahoo! has created a tool called the YUI Compressor that shrinks JavaScript and CSS to the smallest size possible. The resulting files aren't readable by humans, but browsers understand them just fine. You'll work on your files in the human-readable form, but shrink them before putting them on the server. Shrinking these files can save on download time. This shrinking is sometimes referred to as *minifying*.

If you use third-party libraries like jQuery, be sure to deploy the minified versions. Your JavaScript files and style sheets might not be big, but these libraries can be quite large. For example, the regular version of jQuery 1.4.2 is 160k, and the minified version is 70.5k. Most JavaScript libraries can be downloaded in either the regular or minified form.

You can download the YUI Compressor from http://developer.yahoo.com/yui/compressor/.

Google hosts versions of the popular AJAX libraries (like jQuery, Dojo, and YUI) so that you don't have to host them on your own server. This provides a number of advantages. The first is that you don't have to keep your own copy around. The second is that Google's infrastructure speeds up the delivery of these files. And third, if one of your users has already visited a site that is using the Google-hosted version of the file you're using, it's probably already cached so that the browser won't have to download it at all. You can find out how to use Google's copies of the files at http://code.google.com/apis/ajaxlibs/.

Other Good Habits and Hints

In this section, I've gathered several other miscellaneous hints and advice about working with groups of web pages. This includes notes on how big to make each page and how to sign your pages.

Link Back to Home

Consider linking back to the top level or home page on every page of your site. This link will give visitors a quick escape from the depths of your site. Using a home link is much easier than trying to navigate backward through a hierarchy or repeatedly clicking the back button. This is especially important because visitors to most sites are directed there by search engines. If a search engine leads users to an internal page on your site, you'll want to give them a way to find their way to the top.

Don't Split Topics Across Pages

Each web page works best if it covers a single topic in its entirety. Don't split topics across pages; even if you link between them, the transition can be confusing. It will be even more confusing if someone jumps in on the second or third page and wonders what's going on.

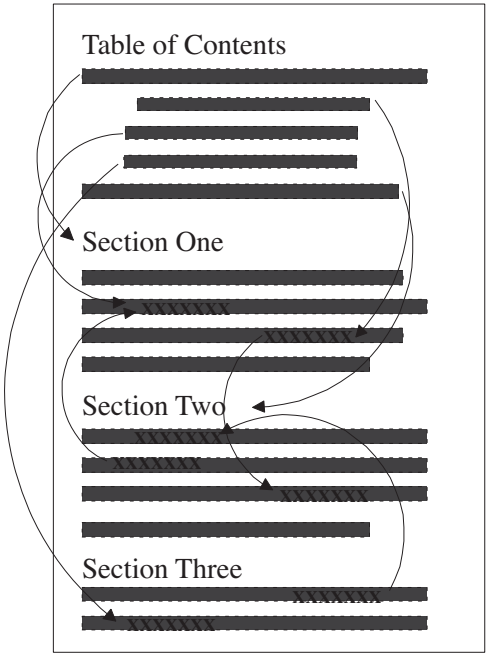
If you think that one topic is becoming too large for a single page, consider reorganizing the page so that you can break up the topic into subtopics. This tip works especially well in hierarchical organizations. It enables you to determine the exact level of detail that each level of the hierarchy should go and exactly how big and complete each page should be.

Don't Create Too Many or Too Few Pages

There are no rules for how many pages your website must have, nor for how large each page should be. You can have one page or several thousand, depending on the amount of content you have and how you've organized it.

With this point in mind, you might decide to go to one extreme or another. Each one has advantages and disadvantages. For example, let's say that you put all your content on one big page and create links to sections within that page, as illustrated in Figure 18.24.

FIGURE 18.24
One big page.



Advantages:

- One file is easier to maintain, and links within that file won't ever break if you move elements around or rename files.
- This file mirrors real-world document structure. If you're distributing documents both online and in hard copy, having a single document for both makes producing them easier.

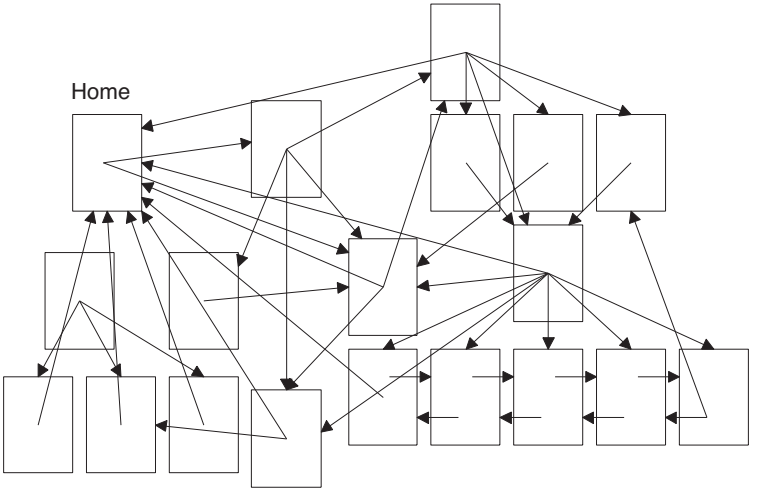
Disadvantages:

- A large file can take a long time to download, particularly if the visitor has a slow network connection and the page includes a large number of images.
- Visitors must scroll a lot to find what they want, and accessing particular bits of information can become tedious. Navigating anywhere other than at the top or bottom becomes close to impossible.

- The structure is overly rigid. A single page is inherently linear. Although visitors can skip around within the page, the structure still mirrors that of the printed page and doesn't take advantage of the flexibility of smaller pages linked in a nonlinear fashion.

At the other extreme, you could create a whole bunch of little pages with links between them, as illustrated in Figure 18.25.

FIGURE 18.25
Many little pages.



Advantages:

- Smaller pages load very quickly.
- You can often fit the entire page on one screen, so the information can be scanned very easily.

Disadvantages:

- Maintaining all those links will be a nightmare. Just adding some sort of navigational structure to that many pages may create thousands of links.
- If you have too many links between pages, the links may seem jarring. Continuity is difficult when your visitors spend more time moving from page to page than actually reading.

What's the solution? The content you're describing will often determine the size and number of pages you need, especially if you follow the one-topic-per-page suggestion. Testing your web pages on a variety of platforms at different network speeds will tell you whether a single page is too large. If you spend a lot of time scrolling around in your page, or if it takes more time to load than you expected, it may be too large.

Sign Your Pages

Each page should contain some sort of information at the bottom to act as the signature. I mentioned this tip briefly in Lesson 7, "Formatting Text with HTML and CSS," as part of the description of the `<address>` tag. That particular tag was intended for just this purpose.

Consider putting the following useful information in the `<address>` tag on each page:

- Contact information for the person who created this web page or who is responsible for it, colloquially known as the *webmaster*. This information should include the person's name and an email address, at the least.
- The status of the page. Is it complete? Is it a work in progress? Is it intentionally left blank?
- The date this page was most recently revised. This information is particularly important for pages that change often. Include a date on each page so that people know how old it is.
- Copyright or trademark information, if it applies.

Figure 18.26 shows a nice example of an address block.

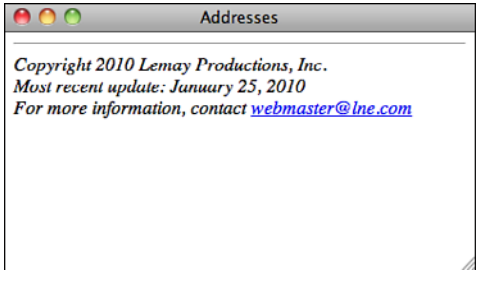


FIGURE 18.26
A sample address.

Another nice touch is to link a Mailto URL to the text containing the email address of the webmaster, as in the following:

```
<address>
Laura Lemay <a href="mailto:lemay@lne.com">lemay@lne.com</a>
</address>
```

This way, the visitors who have browsers that support the Mailto URL can simply select the link and send mail to the person responsible for the page without having to retype the address into their mail programs.

NOTE

One downside of putting your email address on your web page is that there are programs that search websites for email addresses and add them to lists that are sold to spammers. You'll want to consider that risk before posting your email address on a public web page.

Finally, if you don't want to clutter each page with a lot of personal contact information or boilerplate copyright info, a simple solution is to create a separate page for the extra information and then link the signature to that page. Here's an example:

```
<address>
<a href="copyright.html">Copyright</a> and
<a href="webmaster.html">contact</a> information is available.
</address>
```

Summary

The main do's and don'ts for web page design are as follows:

- Do understand the differences between the HTML standards. Decide which design strategy to follow while using them.
- Do provide alternatives if at all possible if you use nonstandard HTML tags.
- Do test your pages in multiple browsers.
- Do write your pages clearly and concisely.
- Do organize the text of your page so that your visitors can scan for important information.

- Do spell check and proofread your pages.
- Do group related information both semantically (through the organization of the content) and visually (by using headings or separating sections with rule lines).
- Do use a consistent layout across all your pages.
- Do use link menus to organize your links for quick scanning, and do use descriptive links.
- Do have good reasons for using links.
- Do keep your layout simple.
- Do provide alternatives to images for text-only browsers.
- Do try to keep your images small so that they load faster over the network.
- Do be careful with backgrounds and colored text to avoid making your pages flashy but unreadable.
- Do use external CSS and JavaScript files whenever possible.
- Do always provide a link back to your home page.
- Do match topics with pages.
- Do provide a signature block or link to contact information at the bottom of each page.
- Do write context-independent pages.
- Don't link to irrelevant material.
- Don't write web pages that are dependent on pages before or after them in the structure.
- Don't overuse emphasis (such as boldface, italic, all caps, link text, blink, or marquees).
- Don't use terminology that's specific to any one browser ("click here," "use the Back button," and so on).
- Don't use heading tags to provide emphasis.
- Don't fall victim to the "here" syndrome with your links.
- Don't link repeatedly to the same site on the same page.
- Don't clutter the page with a large number of pretty but unnecessary images.
- Don't split individual topics across pages.

Workshop

Put on your thinking cap again because it's time for another review. These questions, quizzes, and exercises will remind you about the items that you should (or should not) include on your pages.

Q&A

Q **I've been creating pages and they work when I test them in the browser. Is it really important to validate them?**

A The number of browser is increasing, not decreasing. Popular desktop browsers include Microsoft Internet Explorer, Firefox, Google Chrome, and Apple Safari. Mobile devices with real web browsers are becoming increasingly popular. It's difficult to test your web pages in all the browsers people are using, and making sure that they validate provides a baseline level of assurance that your pages are built correctly and that they'll work in browsers that you haven't personally tested them with.

Q **I'm converting existing documents into web pages. These documents are text heavy and are intended to be read from start to finish instead of being scanned quickly. I can't restructure or redesign the content to better follow the guidelines you've suggested—that's not my job. What can I do?**

A All is not lost. You can still improve the overall presentation of these documents by providing reasonable indexes to the content (summaries, tables of contents pages, subject indexes, and so on) and including standard navigation links. In other words, you can create an easily navigable framework around the documents themselves. This can go a long way toward improving content that's otherwise difficult to read online.

Q **I have a standard signature block that contains my name and email address, revision information for the page, and a couple of lines of copyright information that my company's lawyers insisted on. It's a little imposing, particularly on small pages. Sometimes the signature is bigger than the page itself! How do I integrate it into my site so that it isn't so obtrusive?**

A If your company's lawyers agree, consider putting all your contact and copyright information on a separate page and then linking to it on every page rather than duplicating it every time. This way, your pages won't be overwhelmed by the legal stuff. Also, if the signature changes, you won't have to change it on every single page. Failing that, you can always just reduce the font size for that block and perhaps change the font color to something with less contrast to the background of the page. This indicates to users that they're looking at fine print.

Quiz

1. What are the three flavors of XHTML 1.0, and which of these three accommodates the widest range of markup?
2. What are some ways you can organize your pages so that visitors can scan them more easily?
3. True or false: Headings are useful when you want information to stand out because they make the text large and bold.
4. True or false: You can reduce the download time of an image by using the `width` and `height` attributes of the `<img>` tag to scale down the image.
5. Why does it improve performance to put your CSS in a linked style sheet rather than including it on the page?
6. What are the advantages and disadvantages of creating one big web page versus several smaller ones?

Quiz Answers

1. The three flavors of XHTML 1.0 are Transitional (designed for the widest range of markup, including tags that are deprecated in the standard), Frameset (which includes all tags in the Transitional specification, plus those for framesets), and Strict (for those who want to stick to pure XHTML 1.0 tags and attributes).
2. You can use headings to summarize topics, lists to organize and display information, and link menus for navigation, and you can separate long paragraphs with important information into shorter paragraphs.
3. False. You should use headings as headings and nothing else. You can emphasize text in other ways, or use a graphic to draw attention to an important point.
4. False. When you use the `width` and `height` attributes to make a large image appear smaller on your page, it may reduce the dimensions of the file, but it won't decrease the download time. The visitor still downloads the same image, but the browser just fits it into a smaller space.
5. Putting your CSS in an external file enables the browser to cache the file so that it doesn't have to download the same information as the user moves from one page on the site to another.
6. The advantages of creating one large page are that one file is easier to maintain, the links don't break, and it mirrors real-world document structure. The disadvantages are that it has a longer download time, visitors have to scroll a lot, and the structure is rigid and too linear.

Exercises

1. Try your hand at reworking the example shown in Figure 18.5. Organize the information into a definition list or a table. Make it easy for the visitor to scan for the important points on the page.
2. Try the same with the example shown in Figure 18.7. How can you arrange the information so that it's easier to find the important points and links on the page?

LESSON 19
Designing for the Real World

In previous lessons, you learned about what you should and shouldn't do when you plan your website and design your pages. You also learned about what makes a good or bad website. There's another important factor that you should take into consideration, and that's how to design your pages for the real world.

You've already learned that the real world consists of many different users with many different computer systems who use many different browsers. Some of the things we haven't yet addressed, however, are the many different preferences and experience levels that the visitors to your site will have. By anticipating these real-world needs, you can better judge how you should design your pages. I also explain how you can make sure that your websites are usable for people who are disabled and must use accessibility technologies to browse the Web.

In this lesson, you'll learn some ways that you can anticipate these needs, as well as the following:

- Things to consider when you're trying to determine the preferences of your audience
- Various ways of helping users find their way around your site
- HTML code that displays the same web page in each of the XHTML 1.0 specifications (Transitional, Frameset, and Strict)
- What accessibility is, and how to design accessible sites
- Using an accessibility validator

What Is the Real World, Anyway?

You're probably most familiar with surfing the Internet on a computer that runs a specific operating system, such as Windows, Mac OS X, or something similar. You might think you have a pretty good idea of what web pages look like to everyone.

Throughout this book, you've learned that the view you typically see on the Web isn't the view that everyone else sees. The real world includes many different computers with many different operating systems. Even if you try to design your pages for the most common operating system and the most common browsers, there's another factor that you can't anticipate: *user preference*. Consider the following family, for example:

- Bill is a top-level executive at a Fortune 100 company that has its own intranet. The company IT department requires everyone to use the same operating system and the same browser. Bill is mainly interested in getting news online, and doesn't bother with multimedia. He's also addicted to his smartphone, constantly checking his email and using it to get news on the Web when he's not at his desk.
- Bill's wife, Susan, uses her computer mainly for email. She follows links that she gets from other people, but she's not comfortable "surfing" the Web. She's a genealogist by hobby and has learned that the Internet has many resources in that field. She also wants to publish her family history on the Web. When she and her husband got their cable modem hooked up, she was thrilled. But soon she was asking questions such as, "Can we fit more on the screen? Those letters are a bit too small…can we make them larger? Where are the pictures? How come you have the music turned off? It says that there's sound on this page!" She already wanted to see the Internet much differently than what her husband was used to seeing.
- Bill and Susan have a son, Tom, who's in high school. He's an avid gamer and spends a lot of time watching videos on YouTube! He pumps up the volume as loud as he can and pushes the capacities of their new computer to the max. He also thinks "Browser X" is better than "Browser Y" because it supports lots of cutting-edge features. He wants to start a blog that provides hints, tips, and tricks for one of his favorite online games.
- Tom's older sister, Jill, is an art major in college, studying to be a commercial artist. She has a keen interest in sculpture and photography. She plans to use her new computer for homework assignments, so she'll be looking at the Web with a keen visual interest. She also uses Facebook and Twitter to stay in touch with her friends from school.
- Then there are the senior members of the family, Susan's aging parents, who have recently moved in with the family after years of living in a rural town. Their

experience with computers is minimal—they don't even have their own email addresses. They're interested in learning so that they can view family photos online and exchange email with out-of-town relatives, but Dad's eyes aren't quite as sharp as they used to be. He needs a special browser so that he can hear the text as well as see it.

All these people are using the same computer and operating system to view the Web. In all cases but one (young Tom), they're also using the same browser. This example illustrates one of the other things that you need to think about when you design your website: the needs of the users themselves. Some of these needs are easier to accommodate than others. The following section describes some of the considerations you saw in the previous example.

Considering User Experience Level

There are varied levels of experience in our fictitious family. Although everyone is keenly interested in the Web, some of them have barely used a web browser. When you design your site, consider that the people who visit it might have varying levels of experience and browsing requirements.

Will the topics that you discuss on your site be of interest to people with different levels of experience? If so, you might want to build in some features that help them find their way around more easily. The key, of course, is to make your navigation as intuitive as possible. By keeping your navigation scheme consistent from page to page throughout the site, you'll do a favor for users of all experience levels. There are a number of features you can add to your site that will improve its usability for everyone.

Add a Search Engine

Many users go straight to the search engine when they want to find something on a site. No matter how much time and effort you put into building a clear, obvious navigation scheme, someone looking for information about Frisbees is going to look for a box on your page where they can type in the word *Frisbee* and get back a list of the pages where you talk about them.

Unfortunately, locating a good search engine package and setting it up can be an awful lot of work, and difficult to maintain. On the other hand, there are some alternatives. Some search engines enable you to search a specific site for information. You can add a link to them from your site. Some search engines even allow you to set things up so that you can add their search engine to your site, such as Google:

http://www.google.com/cse/

By signing up, you can add a search box to your site that enables your users to search only pages on your own site for information. For a list of other ways to add search functionality to your site, see the following page in the Open Directory Project:

http://www.dmoz.org/Computers/Software/Internet/Servers/Search/ Use Frames Wisely

One failing of some Hypertext Markup Language (HTML) books is that they tell you about all the different techniques you can use to create web pages, but they don't offer any comparative information that explains when and how these techniques ought to be used. For example, I've devoted a number of pages to discussing frames, but they should really be used only when nothing else will work. For example, putting all of your navigation in one frame and your content in another just because you can is generally a poor idea.

Frames have a couple of specific disadvantages that make them unsuitable for use in many cases. The first is that they make it hard for users to bookmark inner pages on sites. If your entire site is a frameset, the URL in the user's location bar never changes as the user navigates within the site. When the user gets to your inner page that has a wonderful recipe for chocolate chip cookies that your grandmother gave you, he'll have a hard time bookmarking it because his browser wants to bookmark the top-level frameset itself. When he returns on future visits, he'll be taken back to your home page.

The second issue that's common with framesets is that they can interfere with search engines. Again, if your entire site is inside a frameset, when a search engine fetches your home page, it's probably going to get a page with no real content to speak of. Then, it will download, say, the navigation frame, which lacks context, the content frame, which may lack header and footer information, and on and on. If you want people to find your site using search engines, frames can get in your way.

There are also other issues people have had with framesets in the past, like problems printing and problems with the back button. Browsers mostly compensate for these issues today, but even so, frames can confuse your users. In some cases, there's just no other good way to attack a problem, but you should always make sure that there's not a better way before going the frames route.

Use Concise, Sensible URLs

One common mistake made by web designers is not considering how users share URLs. If your site is interesting at all, people are going to email the URL to their friends, paste it into instant messaging conversations, and talk about it around the water cooler. Making your URLs short and easy to remember makes them that much easier for people to share. There's a reason why people have paid huge sums for domain names like `business.com` and `computers.com` in the past. They're easy to remember and you don't have to spell them out when you tell them to people.

You may not have any control over your domain name, but you can exercise control over the rest of your URLs. Say that you have a section of your site called "Products and Services." All the pages in that section are stored in their own directory. You could call it any one of the following:

```
/ps
/prdsvcs
/products
/products_services
/products_and_services
```

There are plenty of other options, too (you could call it `/massapequa` if you wanted to), but the preceding list seems like a reasonable group of options. Of the list, a few stand out to me as being poor choices. `/products_service` and `/products_and_services` just seem too verbose. If the pages under those directories have long names, you're suddenly in long URL territory, which isn't conducive to sharing. On the other hand `/prdsvcs` may be short, but it's also difficult to remember and almost certainly has to be spelled out if you tell it to anyone. It's probably no good. That leaves two remaining choices: `/ps` and `/products`. The first, `/ps`, is nice and short, and probably easy to remember. Using it would be fine. However, there's one other principle of URLs that I want to talk about: guessability.

Chances are that most of the people who visit your website have been using the Web for awhile. There's some chance that they might just assume that they know where to go on your site based on experience. If they want to read about your products, they may guess—based on their experience with other sites they've visited—that your products will be listed at `http://www.your-url.com/products`. Any time you can put your content where your users will assume it to be, you're doing them a favor. Using standard directory names such as `/about`, `/contact`, and `/products` can make things ever so slightly easier for your users at no cost to you.

My final bit of advice on URLs is to make sure that they reflect the structure of your site. One time I worked on a site that consisted of hundreds of files, all in a single directory. The site itself had structure, but the files were not organized based on that structure. Whether the user was on the home page or five levels deep within the site, the URL was still just a filename tagged onto the hostname of the server. Not only did this make the site hard to work on, but it also kept some useful information away from users. Suppose you have a site about cars. What's more useful to your users?

http://www.example.com/camry.html

or

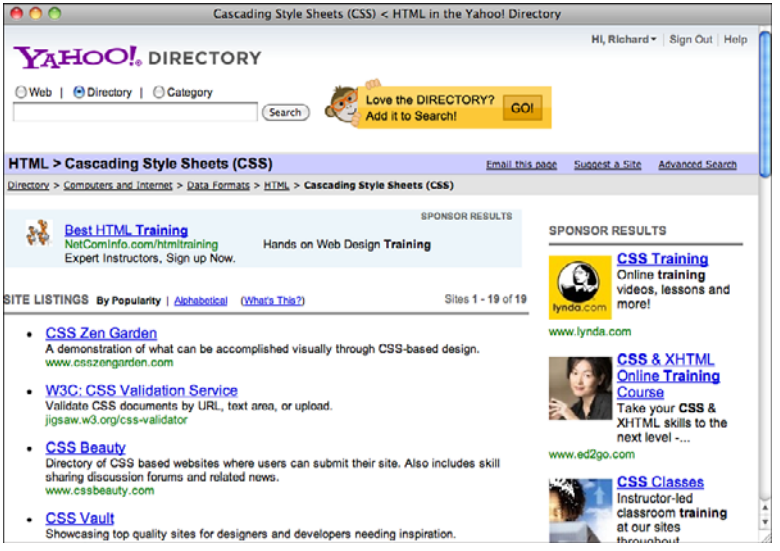
http://www.example.com/cars/toyota/2009/camry.html

The second URL provides a lot more information to the user than the first one does. As an added bonus, you can set up your site so that the user can take `camry.html` off the end and get a list of all Toyota models for 2009, or take `toyota/2009/camry.html` off the end and get a list of all car makes discussed on the site. Veteran web users are accustomed to dealing with URLs; you should help them out as much as you can by using URLs responsibly.

Navigation Provides Context

The key purpose of navigation is obviously to enable your users to get from one place to another within your site. However, its secondary purpose is to let your users know where they are within the site. This was the stroke of genius behind Yahoo!'s introduction of "breadcrumb" navigation in its directory. Take a look at the screenshot from Yahoo!'s directory in Figure 19.1.

FIGURE 19.1
A page from Yahoo!'s directory.



Near the top, you can see a list of links that start at the top of the directory and lead down to the page that I'm actually on. The first thing it does is give me the ability to go back to any level of the directory between the home page and the Cascading Style Sheets page that I'm actually on. The second thing it does is let me know that I'm six levels into the directory, and that the page I'm on is part of the Arts category of the directory, and all the subcategories between that category and the page that I'm on. That's a lot of utility packed into a small feature.

Not all sites have as large and complex a structure as Yahoo!, but you can still provide context for your users through your navigation scheme. By altering your navigational elements based on the page that the user is on, you can indicate to them not only where she can go but also where she is. This is also particularly helpful to users who arrive at your site not via the home page, but from an external link. Enabling users to immediately deduce where they are in the larger scheme of things makes it more likely that they'll take in more of your site.

Are Your Users Tourists or Regulars?

When you're designing a site, especially one that uses forms, one of the key questions you have to ask yourself is whether your users are tourists or regulars. If your users are tourists, which means that they don't use your site very often, or will probably only ever use it one time, you should design your site so that the first-timer can easily figure out what he should be doing and where he needs to go. This may annoy regular users who already know where they're going, but in some cases you have to cater to tourists.

On the other hand, if your site is normally used by the same existing group of users who come back once a day or once a week, your emphasis should be on providing shortcuts and conveniences that enable them to use your site as efficiently as possible. It's okay if it takes a bit of work to learn about the conveniences, because it's worth your users' time.

Clearly, the secret is to strike a balance here. The holy grail is a site that's obvious and clear to new users but also provides the features that repeat users crave. However, understanding what sort of audience you have can help you determine how to assign your resources.

All of this is doubly true with forms. The forms that are part of a discussion board used by the same people day after day will be designed much differently from those that are part of a request-more-information form on a product page. When you design a form, always think about the type of user who'll be using it.

Determining User Preference

In addition to the various levels of experience that visitors have, everyone has his own preferences for how he wants to view your web pages. How do you please them all? The truth is, you can't. But you *can* give it your best shot. Part of good web design is anticipating what visitors want to see on your site. This becomes more difficult if the topics you discuss on your site are of interest to a wider audience.

You'll notice that each person in our fictitious family needs to see the Web differently. Sometimes this is due to his or her interests, but other times it's because of special needs. Therein lies the key to anticipating what you'll need on your web pages.

A topic such as "Timing the Sparkplugs on Your 300cc Motorcycle Engine" is of interest to a more select audience. It will attract only those who are interested in motorcycles—more specifically, those who want to repair their own motorcycles. It should be relatively easy to anticipate the types of things these visitors would like to see on your site. Step-by-step instructions can guide them through each process, while images or multimedia can display techniques that are difficult to describe using text alone.

"The Seven Wonders of the Ancient World," on the other hand, will attract students of all ages as well as their teachers. Archaeologists, historians, and others with an interest in ancient history also might visit the site. Now you have a wider audience, a wider age range, and a wider range of educational levels. It won't be quite as easy to build a site that will please them all.

In cases such as this, it might help to narrow your focus a bit. One way is to design your site for a specific user group, such as the following:

- **Elementary school students and their teachers**—This site requires a basic navigation system that's easy to follow. Content should be basic and easy to read. Bright, colorful images and animations can help keep the attention of young visitors.
- **High school students and their teachers**—You can use a slightly more advanced navigation system. Multimedia and the latest in web technology will keep these students coming back for more.
- **College students and their professors**—A higher level of content is necessary, whereas multimedia may be less important. Properly citing the sources for your information will be important.
- **Professional researchers and historians**—This type of site probably requires pages that are heavier in text content than multimedia.

It's not always possible to define user groups for your website, so you'll need to start with your *own* preferences. Survey other sites that include similar content. As you browse through them, ask yourself what you hope to see there. Is the information displayed well? Is there enough help on the site? Does the site have too much or too little multimedia? If you can get a friend or two to do the survey along with you, it helps you get additional feedback before you start your own site. Take notes and incorporate those ideas into your own web pages.

After you design some initial pages, ask your friends, family members, and associates to browse through your site and pick it apart. Keep in mind that when you ask others for constructive criticism, you might hear some things that you don't want to hear. However, this process is important because you'll often get many new ideas on how to improve your site even more.

Migrating to HTML5

One big question facing web developers right now is whether to start using HTML5 or to stick with XHTML 1.0 (or even HTML 4.01). HTML5 is the path forward for HTML, but it has not quite arrived. Work on it started in 2004, and as of February 2010, it was in last call status with the WHATWG, the group that is drafting the specification. After the specification is finalized, browsers will need to add support for the new features in the specification.

Benefits of HTML5

Why are browser makers and interested publishers spending years coming up with a new version of HTML? In the earlier days of the Web, new features were added to HTML regularly without much regard for the standards process. Frames were created because Netscape thought it would be a good idea to add a feature that allowed publishers to put ads on the page that would move out of sight when users scrolled down the page. Microsoft added the `basefont` tag to HTML so that authors could specify the font for a page in one location before support for CSS was added to browsers. Furthermore, browser makers added support for the same tags in different ways, so HTML looked different depending on the user's browser. HTML5 will contain only features that the makers of all the major browsers have agreed to support, so it allows HTML to move forward without the browser wars that were one of the worst things about the early Web. Having said that, browsers will probably implement the new features in HTML5 on different schedules. Mobile Safari, the browser on Apple's iPhone, already supports a number of the new features in HTML5. Google Chrome, Apple Safari, and Firefox all support HTML5, as does Internet Explorer 9.

HTML5 introduces a number of new tags that more accurately reflect the way pages are built now. Most designers build pages using lots of `<div>` tags with classes and IDs that describe the purpose of those elements. HTML5 adds tags like `<article>`, `<header>`, `<footer>`, `<nav>`, and `<section>`. Browser support for the new elements is limited, but they will be supported eventually.

HTML5 also drops a number of tags that were deprecated in previous HTML specifications. I've noted which tags have been dropped throughout the book. It's unknown whether browsers will ignore those tags if a page uses the HTML5 DOCTYPE, but they are no longer part of HTML5.

One important new feature of HTML5, which I discussed in Lesson 12, "Integrating Multimedia: Sound, Video, and More," is native video playback without plug-ins. The `<video>` tag can be used to embed movies in web pages without the Flash player or any other video player.

Likewise, HTML5 also enables you to create images using markup via the `<canvas>` element. The `<canvas>` tag is used to define the size and location of the generated image, and the actual image is defined using JavaScript. The `<canvas>` element will allow web programmers to do things such as create graphs on-the-fly without using external images. Browsers like Safari, Google, Chrome, and Firefox already support the `<canvas>` element.

In addition to new tags, HTML5 provides some new interfaces for JavaScript programmers. For programmers, HTML5 enables web applications to behave a lot more like desktop applications, incorporating features that have been implemented outside the browser by Flash and other plug-ins, and making some features that developers implement in JavaScript more powerful.

HTML5 supports local storage for web applications, enabling websites to actually store data on the user's computer. This enables applications to manage their own data cache on each computer. They can store images or other web content locally so that it doesn't have to be downloaded each time the user visits the site, or store application the user enters on their own computer. For example, using local storage, you could create a browser-based email client that allows users to draft outgoing emails when the computer is offline, and then send them all when the user visits the application after reconnecting to the Internet.

HTML5 includes drag-and-drop features that can be used to build richer applications. You can already add drag and drop to applications using JavaScript, but it only applies to elements on a single web page. In current browsers, you can create a catalog that enables users to drag items to a shopping cart. With HTML5 drag and drop, you can enable users to drag items from one web page to another, or even drag things from other applications to the browser. For example, you could create a page that enables users to upload files simply by dragging them from the desktop into the browser window, rather than using the form control for files. The new drag-and-drop features have already been implemented in Firefox.

What's Not in HTML5

As mentioned, HTML5 removes a number of tags that have been slated for removal for some time. In most cases, the tags and attributes that were removed are either redundant or have been replaced by CSS properties. For example, the `<acronym>` tag has been removed because it serves the same purpose as the `<abbr>` (abbreviation) tag. The `border` attribute of the `<img>` tag has been removed because you can use the CSS `border` property to achieve the same effect.

However, some tags and attribute have been removed without equivalent replacements. The biggest feature removed by HTML5 is frames. HTML5 supports the `<iframe>` tag, which can be styled using CSS, but `<frameset>`, `<frame>`, and `<noframes>` have all been removed. In the future, if you want to create a framed site in the traditional sense, you'll need to use the HTML 4.01 Frameset or XHTML 1.0 Frameset DOCTYPEs.

Using HTML5

A lot of web designers are eager to dive in and start using all the new features in HTML5. They require fewer browser add-ons, provide new functionality, and enable designers to create simpler, more readable markup. The problem is that not even the most recent browsers offer full support for HTML5, and older browsers don't support it at all. Therefore, if you want a feature to be available to your full audience, even if you use HTML5, you have to provide an alternate accommodation that works for other users.

That said, it's probably time to start migrating to the HTML5 future. This begins with the HTML5 DOCTYPE, which I've used throughout the book for pages that consist of valid HTML5. It is

```
<!DOCTYPE html>
```


Here's the DOCTYPE for HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

As you can see, one of the charms of the HTML5 DOCTYPE is that it's short. It may be the first DOCTYPE declaration that's easy to type in from memory. All current browsers recognize the HTML5 DOCTYPE and render pages that use it in their standards-compliant mode, rather than quirks modes that mimic older, incorrect implementations of standards in old browsers.

The only catch is that if you're going to use the HTML5 DOCTYPE, be sure to avoid using any tags or attributes that have been removed from HTML5.

This is the other big step you should take toward HTML5. New web pages should use CSS to modify the appearance of web pages rather than HTML. Sticking with only those tags and attributes in HTML5 and using CSS for everything else is a big step along that path.

HTML5 and XML

One of the biggest changes from XHTML 1.0 to HTML5 is that unlike XHTML 1.0, HTML5 does not require documents to be valid XML. In XHTML 1.0, every tag had to be closed, and every attribute was required to have a value to be valid XML documents. HTML5 is like HTML 4.01—closing tags and attribute values are not strictly required.

That said, HTML5 documents can be formatted using valid XML—that's the approach I've taken in this book. Either approach is acceptable and browsers support both.

To indicate to the browser whether the page is valid XML, the `Content-type` header is used. When a browser requests a web page, or an image, or any other resource from a server, the response from the server includes a number of headers. These headers are used by the server and browser to exchange information that isn't part of the web page.

The `Content-type` header is what the server uses to tell the browser what kind of resource it's sending. The content types themselves come in two parts, separated by a slash. The part before the slash describes the general type of the file, and the part after the slash identifies the specific format of the file. So when a JPEG image is sent to the browser, the content type is `image/jpeg`. Regular web pages use the content type `text/html`. The content type enables the browser to figure out what to do with the resource. Most of the time, it displays it in the browser, but if the content type is `application/pdf`, it knows it needs to open a PDF reader and use it to display the file.

The HTML5 specification distinguishes between HTML5 that is valid XML and HTML5 that isn't using content types. The content type for vanilla HTML is `text/html`. If you want the browser to treat the document as valid XML, you can use `application/xml` or `application/xhtml+xml`. Right now, however, Microsoft Internet Explorer does not support those content types, so you should serve your HTML5 documents as `text/html`, whether they're valid XML.

Ultimately, it's up to you whether you choose to write XML-compliant HTML. Many web designers are accustomed to XHTML 1.0 and write valid XML, and many tools do a better job of catching errors in valid XML. On the other hand, if you're starting from scratch, you can do what you like.

XHTML 1.0 and HTML 4.01

Most web pages that exist today were created to conform to the XHTML 1.0 or HTML 4.01 specifications. Many of them are not actually valid but work fine in the popular browsers. So if you're working on an existing website, chances are you'll be working with pages in one of these two formats.

XHTML defines HTML so that it conforms to the XML specification. When you want to know whether a particular tag or attribute is valid, check out the HTML specification. When you want to know how to close a tag that has no closing tag, such as `<br>`, consult with the XHTML spec. In earlier lessons, you learned about the various flavors of HTML 4.01 and XHTML 1.0 and how each of them is geared toward users of older or newer browsers:

- **HTML 4.01 or XHTML 1.0 Transitional**—For those who want to provide support for older browsers
- **HTML 4.01 or XHTML 1.0 Frameset**—For sites that want to use frames in a standards-compliant manner
- **HTML 4.01 or XHTML 1.0 Strict**—For those who want to develop pages that strictly adhere to the HTML 4.0 or XHTML 1.0 specification by not using deprecated elements or attributes

You may encounter these three DOCTYPEs when you work on existing pages, or if you want to create a site that uses frames, you'll need to use one of the two frameset DOCTYPEs.

What Is Accessibility?

Accessibility is basically the effort to make websites as usable as possible for people with disabilities. This involves the creation of software and hardware that enables people with various disabilities to use computers and the Web. It also means addressing accessibility concerns in the design of HTML as a markup language and efforts on the part of web designers to incorporate accessibility into their websites. When a person with impaired vision uses a screen reader to visit a website, there are things the site's author can do to make that experience as rich and fulfilling as possible given the user's disability.

Common Myths Regarding Accessibility

Historically, there has been some resistance among web designers toward building websites in an accessible manner. This resistance has arisen not due to a want to discriminate against people who might benefit from accessible design, but rather from a fear that

accessibility will limit designers' options in how they create their sites. Also accessibility seems like it will add additional work, and most people have too much to do already.

For a long time, many people thought that *accessible* was a code word for *all text*. It was believed that adding accessibility meant putting all your content in a single column running straight down the page and avoiding the bells and whistles that many people believe are necessary for an attractive website. This couldn't be further from the truth. Although some common techniques can interfere with accessibility, that doesn't mean that you must remove any images, sounds, or multimedia from your website. Nor does it dictate that your layout be simplified.

The demand that accessibility places on designers is that they write clean, standards-compliant markup, take advantage of HTML features that improve accessibility, and use tags as they are intended to be used in the specification rather than based on how they make your pages look in the browser. The best example here is tables. At one time, nearly all websites used tables not only for showing things like tables of statistics, but also for controlling the layout of entire pages. These types of approaches run counter to how HTML was intended to be used and make things much more difficult for users with alternative browsers.

To continue to use complex layouts in an accessible world, you have to upgrade to current techniques—in other words, create your layouts using CSS. Just as this approach provides a cutting-edge look and feel in the latest browsers and yet gracefully degrades to still display information adequately in older browsers, it provides the same benefits in alternative browsers. Because the markup is so simple and is properly used when you use CSS to handle layout, alternative browsers for the disabled can handle the markup just fine. That's not the case when you use eight levels of nested tables to make your page look the way you want it to.

The other common misapprehension for accessibility is that it will require a lot of extra work on your part. It does require some extra work—creating your pages so that they take advantage of accessibility features in HTML is more work than leaving them out. However, in many cases, coding for accessibility will help all your users, not just those using alternative browsers.

Section 508

Section 508 is a government regulation specifying that U.S. federal government agencies must provide access for all users, including those with disabilities, to electronic and

information technology resources. It requires that federal agencies consider the needs of disabled users when they spend money on computer equipment or other computer resources. What this boils down to is that federal websites must be designed in an accessible fashion.

Not only did Section 508 change the rules of the game for many web designers (anyone involved with federal websites), but it also raised the profile of accessibility in general. Thanks in part that people didn't understand the implications of Section 508 at first, people started thinking a lot about accessibility and what it meant for the Web.

NOTE

For more information on Section 508, see http://www.section508.gov/.

Alternative Browsers

Just as there are a number of disabilities that can make it more challenging for people to use the Web, there are a number of browsers and assistive technologies that are designed to level the playing field to a certain degree. I discuss some common types of assistive technologies here so that when you design your web pages you can consider how they'll be used by people with disabilities.

Disabled users access the Web in a variety of ways, depending on their degree and type of disability. For example, some users just need to use extra large fonts on their computer, whereas others require a completely different interface from the standard used by most people.

Let's look at some of the kinds of browsers specifically designed for disabled users. For users who read Braille, a number of browsers provide Braille output. Screen readers are also common. Instead of displaying the page on the screen (or in addition to displaying it), screen readers attempt to organize the contents of a page in a linear fashion and use a voice synthesizer to speak the page's contents. Some browsers also accept audio input—users who are uncomfortable using a mouse and keyboard can use speech recognition to navigate the Web.

Another common type of assistive technology is a screen magnifier. Screen magnifiers enlarge the section of the screen where the user is working to make it easier for users with vision problems to use the computer.

Writing Accessible HTML

When it comes to writing accessible HTML, there are two steps to follow. The first step is to use the same tags you normally use as they were intended. The second step is to take advantage of HTML features specifically designed to improve accessibility. I've already mentioned a number of times that tags should be used based on their semantic meaning rather than how they're rendered in browsers. For example, if you want to print some bold text in a standard size font, `<h4>` will work, except that it not only boldfaces the text, it also indicates that it's a level 4 heading. In screen readers or other alternative browsers, that might cause confusion for your users.

Tables

This problem is particularly acute when it comes to tables. I've already mentioned that it's not a good idea to use tables for page layout when you're designing for accessibility. Alternative browsers must generally indicate to users that a table has been encountered, and then unwind the tables so that the information can be presented to the user in a linear fashion. To make things easier on these users, you should use tables as intended where you can. Even if you can't avoid using tables to lay out your page, be aware of how the table will be presented to users. If possible, try to avoid putting lots of navigation text and other supplemental text between the beginning of the page and the content the user is actually looking for.

When you're presenting real tabular data, it's worthwhile to use all the supplemental tags for tables that are all too often ignored. When you're inserting row and column headings, use the `<th>` tag. If the default alignment or text presentation is not to your liking, use CSS to modify it. Some browsers will indicate to users that the table headings are distinct from the table data. Furthermore, if you label your table, using the `<caption>` tag is a better choice than just inserting a paragraph of text before or after the table. Some browsers indicate that the text is a table caption.

Finally, using the `summary` attribute of the `<table>` tag can be a great aid to users with alternative browsers. How you use the summary and caption are up to you. The caption can explain what the table contains, and the summary can explain how the data in the table is used if it's not obvious from the row/column headings. Here's an example of a table that's designed for accessibility:

```
<table summary="This is the famous Boston Consulting Group Product
Portfolio Matrix. It's a two by two matrix with labels." border="1"
cellpadding="12">
  <caption>Boston Consulting Group Product Portfolio Matrix</caption>
  <tr>
    <td colspan="2" rowspan="2"><br /></td>
```

```
    <th colspan="2">Market Share</th>
  </tr>
  <tr>
    <th>High</th>
    <th>Low</th>
  </tr>
  <tr>
    <th rowspan="2">Market Growth</th>
    <th>High</th>
    <td align="center">Star</td>
    <td align="center">Problem Child</td>
  </tr>
  <tr>
    <th>Low</th>
    <td align="center">Cash Cow</td>
    <td align="center">Dog</td>
  </tr>
</table>
```

Links

As mentioned in Lesson 18, "Writing Good Web Pages: Do's and Don'ts," avoiding the "here" syndrome is imperative, particularly when it comes to accessibility. Having all the links on your page described as "click here" or "here" isn't helpful to disabled users (or any others). Just thinking carefully about the text you place inside a link to make it descriptive of the link destination is a good start.

To make your links even more usable, you can use the `title` attribute. The `title` attribute is used to associate some descriptive text with a link. It is used not only by alternative browsers, but many standard browsers will display a tool tip with the link title when the user holds her mouse pointer over it. Here are some examples:


```
<a href="http://www.dmoz.org/"
title="The volunteer maintained directory.">dmoz</a>

<a href="document.pdf" title="1.5 meg PDF document">Special Report</a>
```

Navigational links are a special case because they usually come in sizable groups. Many pages have a nice navigation bar right across the top that's useful to regular users who can skim the page and go directly to the content that they want. Users who use screen readers with their browsers and other assistive technologies aren't so lucky. You can imagine what it would be like to visit a site that has 10 navigational links across the top of the page if you relied on every page being read to you. Every time you move from one page to the next, the navigation links would have to be read over again.

There are a few ways around this that vary in elegance. If you're using CSS to position elements on your page, it can make sense to place the navigational elements after your main content in your HTML file, but use CSS to position them wherever you like. When a user with a screen reader visits the site, he'll get the content before getting the navigation. You can then include a link that skips to the navigation at the top of the page, and hide it using CSS. Users with screen readers can jump to the navigation if they need to but won't be required to listen to it on every page.

TIP

It's worth remembering that many disabled users rely on keyboards to access the Web. You can make things easier on them by using the `accesskey` and `tabindex` attributes of the `<a>` tag to enable them to step through the links on your page in a logical order. This proves particularly useful if you also include forms on your page. For example, if you have a form that has links interspersed in the form, setting up the `tabindex` order so that the user can tab through the form completely before he runs into any links can save him a lot of aggravation. This is the sort of convenience that all of your users will appreciate, too.

Images

Images are a sticky point when it comes to accessibility. Users with impaired vision might not appreciate your images. However, clever design and usage of the tools provided by HTML can, to a certain degree, minimize the problem of image usage.

Images are known for having probably the best-known accessibility feature of any HTML element. The `alt` attribute has been around as long as the `<img>` tag and provides text that can stand in for an image if the user has a text-only browser or the image weren't downloaded for some reason. Back when everybody used slow dialup connections to the Internet, it was easy to become intimately familiar with `alt` text because it displayed while the images on a page downloaded. Later, some browsers started showing `alt` text as a tool tip when the user let her mouse pointer hover over an image.

Despite that `alt` text is useful, easy to add, and required by the HTML5 specification, many pages on the Internet still lack meaningful alternative text for most (if not all) of their images. Taking a few extra minutes to enter `alt` text for your images is a must for anyone who uses HTML that includes images. Also bear in mind that using `alt=""` is

perfectly valid and is sometimes appropriate. In particular, some sites use small images to help position elements on a page. Although this practice is strongly discouraged, it's still used. Text-based browsers will, in the absence of `alt` text, generally display something like `[IMAGE]` on the page. If you include the `alt` attribute but leave it empty, the label `IMAGE` will be left out, making things easier on your users.

HTML also provides another way of describing images that's meant to improve accessibility: the `longdesc` attribute of the `<img>` tag. The `longdesc` attribute is intended to be used as a place to enter long descriptions of images. For example, if an image is a chart or graph, you can explain how it is used and what information it is intended to convey. If the picture is a nice photo of a waterfall, you can describe the waterfall. In many cases, images are used simply to make web pages flashier or aesthetically pleasing, and nothing more than `alt` text is required. However, when the actual meaning of a page is conveyed through the use of images, providing descriptions of those images is a key to accessibility. For example, suppose that you're working for a financial publication, and a story on the declining stock market includes a graph showing the consistent decline of the major indexes over the past few years. In print, that image would probably include a caption explaining what's in the image. On the Web, you could put that caption in the image's `longdesc` attribute. Or if you prefer, you could put it on the page as a caption, as long as your layout indicates in some way that the caption refers to the image in question.

Here's an example of what I'm talking about:

```
<img src="graph1.gif" longdesc="Graph showing that the S&P
500 and Dow Jones Industrial Average have dropped an average of 20%
annually over the last 3 years." />
```


NOTE The `longdesc` attribute has been removed from HTML5. The specification instead recommends that the page include a link to a long description for an image if appropriate.

There's one final area to discuss when it comes to images: the marriage of images and links in the form of image maps. As you might imagine, image maps can be an accessibility issue. Some browsers will display the links from an image map as a menu, but the only certain way around this is to provide users with an alternative to image maps in the form of text links. Whenever you use an image map, make sure to include text equivalents of the links somewhere on the page.

Designing for Accessibility

Just as important as taking advantage of the HTML features provided specifically for accessibility is taking care to design your pages in a manner that's as accommodating as possible for users who are in need of assistance. Most of these techniques are relevant to all users, not just those using alternative browsers or assistive technologies.

Use Color

A common pitfall designers fall into is using color to confer meaning to users. For example, they print an error on the page and change the font color to red to indicate that something went wrong. Unfortunately, visually impaired users can't distinguish your error message from the rest of the text on the page without reading it. Needless to say, putting two elements on the page that are the same except for color (such as using colors to indicate the status of something) is not accessible. You can add borders to elements that need to stand out, or you can label them with text. For example, you might display an error message this way:

```
<p class="error">ERROR: You must enter your full name.</p>
```

Fonts

When you specify fonts on your pages, you can cause accessibility problems if you're not careful. In some cases, font specification doesn't matter at all because the user accesses your site with a screen reader or alternative browser that completely ignores your font settings. However, users who simply see poorly can have an unpleasant experience if you set your fonts to an absolute size—particularly if you choose a small size. If a user has set his browser's default font to be larger than normal, and your pages are hard coded to use 9-point text, that user will probably dump your site altogether.

In many cases, it makes sense to leave the default font specification alone for most of the text on your site. That way, users can set their fonts as they choose, and you won't interfere with their personal preferences. If you do modify the fonts on the page, make sure that the fonts scale with the user's settings so that the user can see the text at a comfortable size.

CAUTION

Be sure to test your pages with a variety of text size settings when you do browser testing. Many users change increase the size of fonts in the browser to make them easier to read, and you should make sure that if users have done so, your pages still work for them.

Take Advantage of All HTML Tags

It's easy to fall into the trap of using `<i>` or `<b>` rather than more specific tags when you need to add emphasis to something. For example, suppose you're citing a passage from a book. When you enter the book title, you could indicate to your users that it's a proper title by putting it inside the `<i>` tag, or you could use the `<cite>` tag. There are plenty of other underutilized tags, as well, all of which provide some semantic meaning in addition to the text formatting they're associated with.

Even in cases in which you really just want to emphasize text, it's preferable to use `<em>` and `<strong>` over `<i>` and `<b>`. These tags provide a lot more meaning than the basic text formatting tags that are often used. Not all alternative browsers will take advantage of any or all these tags, but conveying as much meaning as possible through your choice of tags won't hurt accessibility for sure and will help some now and could help more in the future. There's no downside to taking this approach, either.

Frames and Linked Windows

Frames are, to put it bluntly, a barrier to accessibility. There are some workarounds available, but the bottom line is that if you're concerned about accessibility, you should probably avoid frames. Using linked windows and pop-up windows can also be a huge hassle from an accessibility perspective.

If you opt to use frames, you should include titles for all your frames, not just the document containing the frameset. Using regular browsers, the titles of these documents are suppressed. That's not necessarily the case with alternative browsers. Some will provide links to the individual frames when you pull up the frameset, and having titles makes it easier for users to distinguish between them.

Forms

Forms present another thorny accessibility issue. Nearly all web applications are based on forms, and failure to make them accessible can cost you users. For example, large online stores have a serious financial interest in focusing on form accessibility. How many sales would Amazon or eBay lose if their sites weren't accessible? Some work on making sure the forms that enable you to purchase items are accessible can pay off.

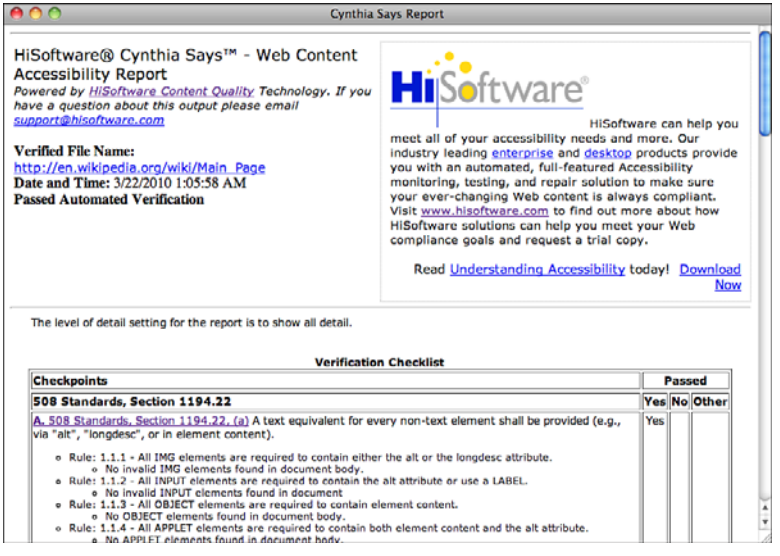
One key thing to remember is that disabled users often navigate using only the keyboard. As I mentioned when talking about links, assigning sensible `tabindex` values to your form fields can really increase both the usability and accessibility of your forms. The other advanced form tags, such as `fieldset`, `optgroup`, and `label`, can be beneficial in terms of usability, too.

Validating Your Sites for Accessibility

There's no reason to rely on luck when it comes to determining whether your site measures up when it comes to accessibility. Just as you can use the W3C validator to verify that your HTML files are standards-compliant, you can use a number of validators to check your site for accessibility problems. Cynthia Says is one such validator, and you can find it at http://www.contentquality.com/. It can validate a site against the Section 508 guidelines mentioned earlier or against the Web Content Accessibility Guidelines developed by the W3C.

Its operation is nearly identical to that of the HTML validator provided by the W3C. If you submit your page to the validator, it generates a report that indicates which areas of your page need improvement, and it provides general tips that can be applied to any page. Figure 19.2 shows a Cynthia Says report for Wikipedia.

FIGURE 19.2
An accessibility report generated by Cynthia Says.



Cynthia Says also links to the relevant standards in the validation results. In Figure 19.3, you can see the destination of one of the links, which points to a passage in the Web Content Accessibility Guidelines.

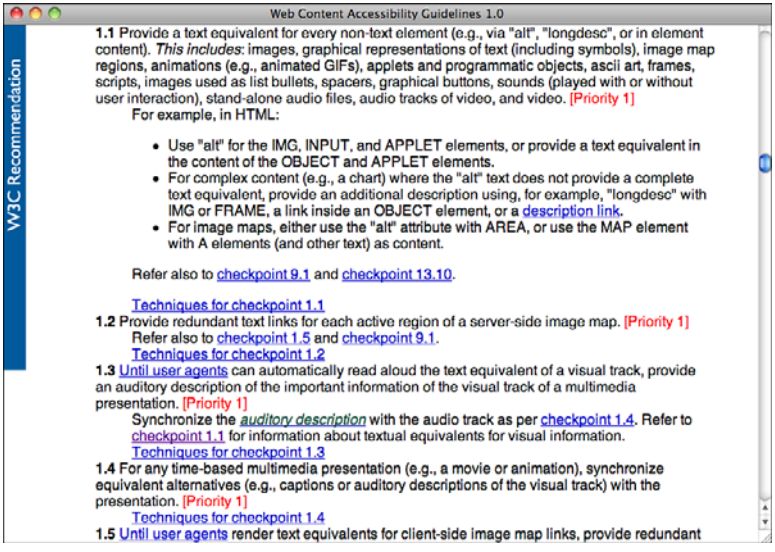


FIGURE 19.3
The Web Content Accessibility Guidelines.

Cynthia Says is provided by HiSoftware, which produces software that helps manage compliance issues for web content. They support Section 508 compliance among other standards and regulations.

Further Reading

This lesson is the tip of the iceberg when it comes to handling accessibility on websites. If you're going to make a commitment to creating an accessible site, you'll probably want to research the issue further. Your first stop should be online accessibility resources. The W3C provides a huge body of information on accessibility as part of its Web Accessibility Initiative. The home page is http://www.w3.org/WAI/.

If you maintain a personal site, you might also find Mark Pilgrim's online book, *Dive into Accessibility* (http://diveintoaccessibility.org/), to be a useful resource. Plenty of other sites discuss accessibility, too. Just enter the word *accessibility* into your favorite search engine to get a list.

There have also been several books written on web accessibility. Joe Clark's *Building Accessible Websites* is well regarded. You can find out more about the book at the book's website: http://joeclark.org/book/.

Summary

The Web is in a period of transition, from the almost universal adoption of XHTML 1.0 toward the new HTML standard: HTML5. In this lesson, I discussed how you can take a progressive approach, moving toward HTML5 without leaving people using current browsers behind.

I hope you now realize that the needs of your visitors should affect the approach you use in your website design. The key is to anticipate those needs and try to address them as broadly as possible. Not every site has to be filled with multimedia that implements the latest and greatest web technologies. On the other hand, certain topics almost demand higher levels of page design. Listen to the needs of your visitors when you design your pages, and you'll keep them coming back.

Even though accessibility issues ostensibly affect only a small percentage of web users, they should not be ignored. Many accessibility-related improvements actually improve the web experience for most users. Leaving out disabled users by not accounting for them in your designs is inconsiderate and can often be a poor business decision, too. Adding accessibility features to an existing site can be challenging, but when you build new sites from scratch, making them accessible can often be done with little additional effort. If I've convinced you of the importance of accessibility in this lesson, you'll probably want to dig into the resources listed previously for more information.

Workshop

As if you haven't had enough already, here's a refresher course. As always, there are questions, quizzes, and exercises that will help you remember some of the most important points in this lesson.

Q&A

Q **Feedback from visitors to my site varies a lot. Some want my pages to use less multimedia, whereas others want more. Is there an easy way to satisfy both of them?**

A You've already learned that you can provide links to external multimedia files. This is the best approach for visitors who want less multimedia because they won't see it unless they click the link.

You can also simply ask them which version of your site they want to see. I generally recommend building a site that works well for users regardless of their connection speed or browser capabilities, but in some cases it makes sense to create alternative versions of your site. You can start out with an entry page that allows

your users to choose between the different versions of the site, or you can start out with the fancier page and provide a link to the text version that shows up regardless of their browser's capabilities.

Q I use a lot of external files on my website, and they can be downloaded from several different pages. Wouldn't it be more efficient to include a link to the correct readers or viewers on the pages where the external files appear?

A Although it's much easier for the visitor to download an external file and the appropriate reader or helper application from the same page, it might be more difficult for you to maintain your pages when the URLs for the helper applications change. A good compromise is to include a Download page on your website with links to all the helper applications that the visitor will need. After the visitor downloads the external file, she can then navigate to your Download page to get the helper application she needs to view that file.

Q If I don't make my site accessible, what percentage of my audience will I lose?

A Even if you weren't wondering about this, there's a good chance your boss probably wants to know. Unfortunately, there's no hard-and-fast number. I've seen it reported that 10% of the population has disabilities, but not all those disabilities affect one's ability to access the Web.

Q Can I run into legal trouble if I don't bother with making my site accessible?

A If you're in the United States, the answer to this question is no, unless you're working on a site for the federal government and are bound by Section 508.

Quiz

1. How do real-world user needs vary?
2. What are some important things to include on your site to help those who are new to computers or the Internet?
3. What are two things you can do right now to start migrating to HTML5?
4. True or false: It's better to have a lot of frames in a frameset because you can keep more information in the browser window at the same time.
5. True or false: To make a site truly accessible, no images can be used for navigation or links.
6. How should navigation be placed on a page to make it most accessible?
7. Name three attributes of tags aimed specifically at accessibility.

Quiz Answers

1. Different users will have different levels of experience. Browser preferences will vary. Some want to see a lot of multimedia, whereas others prefer none at all. Some prefer images and multimedia that are interactive, whereas others prefer simpler pictures that demonstrate a process or technique on how to do something. Other preferences are more specific to the interests of the visitors.
2. Include pages on your site that help tvisitors find the information they're looking for. Also include pages that help them find their way around the site.
3. Two things you can do to start migrating to HTML5 are to start using the HTML5 DOCTYPE and to stop using the tags and attributes that were removed from HTML5 in your pages.
4. False. Too many frames can be confusing for new users, and they might be too small to be useful when they're viewed at lower resolutions.
5. False; however, you must use the images in an accessible manner.
6. Navigation should be placed after the main content on a page to make it accessible with users who must navigate the page in a linear fashion.
7. Some attributes designed to improve accessibility are the `title` attribute of the `<a>` tag, the `longdesc` and `alt` attributes of the `<img>` tag, and the `summary` attribute of the `<table>` tag. Remember, though, that `longdesc` and `summary` have been removed from HTML5. It is recommended that their content be incorporated into the page in another way.

Exercises

1. Design a simple navigation system for a website and describe it in a manner that makes sense to you. Then ask others to review it and verify that your explanations are clear to them.
2. Make a list of the topics that you want to discuss on your website. Go through the list a second time and see whether you can anticipate the types of people who will be interested in those topics. Finally, review the list a third time and list the special needs that you might need to consider for each user group.
3. Visit Cynthia Says, the accessibility validator, and see how your site rates against the accessibility guidelines.
4. Make sure that all the `<img>` tags on your site have `alt` attributes. It's a good first step toward accessibility.

LESSON 20

Putting Your Site Online

Just uploading your site to a web server somewhere doesn't mean that you'll attract many visitors. With millions of sites online already, you'll need to promote your site if you want to build an audience.

So, how do you entice people to come to your site? This lesson shows you some of the ways, including the following:

- What a web server does and why you need one
- How to find web hosting
- How to deploy your website
- How to find out your URL
- How to test and troubleshoot your web pages
- Advertising your site
- Submitting your site to search engines
- Using business cards, letterheads, and brochures
- Promoting your site on social networks
- Using analytics to find out who's viewing your pages

What Does a Web Server Do?

To publish web pages, you need a web server. The server listens for requests from web browsers and returns the resources specified in the URL in those requests. Web servers and web browsers communicate using the Hypertext Transfer Protocol (HTTP), a protocol created specifically for the request and transfer of hypertext documents over the Web. Because of this use, web servers often are called *HTTP servers*.

Other Things Web Servers Do

Although the web server's primary purpose is to answer requests from browsers, it's responsible for several other tasks. You'll learn about some of them in the following sections.

File and Media Type Determination

In Lesson 12, "Integrating Multimedia: Sound, Video, and More," you learned about content types and how browsers and servers use file extensions to determine file types. Servers are responsible for telling the browsers what kinds of content the files contain. Web servers are configured so that they know which media types to assign to files that are requested so that the browser can tell audio files from HTML pages from style sheets.

File Management

The web server also is responsible for rudimentary file management—mostly in determining how to translate URLs into the locations of files on the server. If a browser requests a file that doesn't exist, the web server returns the HTTP error code 404 and sends an error page to the browser. You can configure the web server to redirect from one URL to another, automatically pointing the browser to a new location if resources move or if you want to retire them. Servers can also be set up to return a particular file if a URL refers to a directory on a server without specifying a filename.

Finally, servers keep log files for information on how many times each URL on the site has been accessed, including the address of the computer that accessed it, the date and, optionally, which browser they used, and the URL of the page that referred them to your page. Web servers also keep a log of any errors that occur when browsers submit requests so that you can track them down and fix them.

Server-Side Scripts and Forms Processing

In addition to serving up static documents such as HTML files and images, most web servers offer the option of running scripts or programs that generate documents on-the-fly. These scripts can be used to create catalogs and shopping carts, discussion boards, clients to read email, or content management systems to publish documents dynamically.

Any website that you find that does more than just publish plain old documents is running some kind of script or program on the server. A number of popular scripting platforms are available for writing web applications. Which one is available for your use depends in part on which web server you're using. PHP is probably the most popular choice. It's easy to get started with and runs on most servers. Other popular choices include Microsoft .NET, which runs on Windows, or Java Server Pages (JSP), which can run on most servers. Newer choices include Ruby on Rails and Django, both of which are frameworks that can be used to build web applications.

Server-Side File Processing

Some servers can process files before they send them along to the browsers. On a simple level, there are server-side includes, which can insert a date or a chunk of boilerplate text into each page, or run a program. Also, you can use server-side processing in much more sophisticated ways to modify files on-the-fly for different browsers or to execute small bits of code embedded in your pages.

Authentication and Security

Password protection is provided out-of-the-box by most web servers. Using authentication, you can create users and assign passwords to them, and you can restrict access to certain files and directories. You can also restrict access to files or to an entire site based on site names or IP addresses. For example, you can prevent anyone outside your company from viewing files that are intended for employees. It's common for people to build custom authentication systems using server-side scripts, too.

For security, some servers also provide a mechanism for encrypted connections and transactions using the Secure Sockets Layer (SSL) protocol. SSL allows the browser to authenticate the server, proving that the server is who it says it is, and an encrypted connection between the browser and the server so that sensitive information between the two cannot be understood if it is intercepted.

How to Find Web Hosting

Before you can put your site on the Web, you must find a web server. How easy this is depends on how you get your access to the Internet.

Using a Web Server Provided by Your School or Work

If you get your Internet connection through school or work, that organization might allow you to publish web pages on its own web server. Given that these organizations usually have fast connections to the Internet and people to administer the site for you, this situation is ideal.

You'll have to ask your system administrator, computer consultant, webmaster, or network provider whether a web server is available and, if so, what the procedures are for putting up your pages. You'll learn more about what to ask later in this lesson.

Using a Commercial Web Host

You may pay for your Internet access through an Internet service provider (ISP), or a commercial online service. Many of these services allow you to publish your web pages, although it may cost you extra. Restrictions might apply as to the kinds of pages you can publish or whether you can run server-side scripts. You can probably find out more about the web hosting options offered by your Internet service provider on the support section of its website. Many companies that specialize in web hosting have popped up. These services, most commonly known as *web hosts*, usually provide a way for you to transfer your files to their server (usually FTP or secure FTP), as well as the disk space and the actual web server software that provides access to your files. They also have professional systems administrators onsite to make sure the servers are running well at all times.

Generally, you're charged a flat monthly rate, with added charges if you use too much disk space or network bandwidth. Many web hosts provide support for server-side scripts written in PHP and often install some commonly used scripts so that you don't even have to set them up. Most also enable you to set up your site with your own domain name, and some even provide a facility for registering domain names, too. These features can make using commercial web hosting providers an especially attractive option.

CAUTION

> Make sure that when you register your domains, they are registered in your name rather than in the name of the hosting provider or domain registrar who registers them on your behalf. You want to make sure that you own the domain names you register.

To get your own domain name, you need to register it with an authorized registrar. The initial cost to register and acquire your domain name can be as low as $8 per year. Thereafter, an annual fee keeps your domain name active. After you have your own domain name, you can set it up at your hosting provider so that you can use it in your URLs and receive email at that domain. Your site will have an address such as http://www.example.com/.

Many ISPs and web hosts can assist you in registering your domain name. You can register your domain directly with an authorized registrar such as Network Solutions

(http://www.networksolutions.com/), Register.com, dotster.com, or godaddy.com. Most of these services also offer *domain parking*, a service that allows you to host your domain with them temporarily until you choose a hosting provider or set up your own server. The prices vary, so shop around before registering your domain.

Setting Up Your Own Server

If you're really courageous and want the ultimate in web publishing, running your own website is the way to go. You can publish as much as you want and include any kind of content you want. You'll also be able to use forms, scripts, streaming multimedia, and other options that aren't available to people who don't have their own servers. Other web hosts might not let you use these kinds of features. However, running a server definitely isn't for everyone.

There are two options here. The first is to set up an actual computer of your own and use it as a server. However, the cost and maintenance time can be daunting, and you need a level of technical expertise that the average user might not possess. Furthermore, you need some way to connect it to the Internet. Many Internet service providers won't let you run servers over your connection, and putting your server in a hosting facility or getting a full-time Internet connection for your server can be costly. However, this might be the right answer if you are setting up a website for internal use at your company or organization.

The second option is to lease a virtual server. Applications exist that enable companies to treat a single computer as multiple virtual computers. They then lease those virtual computers to people to use for whatever they like. So for a modest price, you can lease a virtual server over which you have full control. From your perspective, it is your computer. Companies such as Slicehost (http://slicehost.com) and Linode (http://linode.com) offer virtual servers, as does Amazon.com through their EC2 service.

Free Hosting

If you can't afford to pay a web hosting provider to host your website, some free alternatives exist. For the most part, free sites do not offer the opportunity to create your own pages by hand and deploy them. Instead, there are services that host particular kinds of content like weblogs (http://www.blogger.com/), journals (http://www.livejournal.com/), or photos (http://www.flickr.com/). These are just some examples. The trade-off is that the pages on these sites have advertisements included on them and that your bandwidth usage is generally sharply limited. There are often other rules regarding the amount of space you can use, too. Free hosting can be a good option for hobbyists, but if you're serious about your site, you'll probably want to host it with a commercial service.

Organizing Your HTML Files for Publishing

After you have access to a web server, you can publish the website you've labored so hard to create. Before you actually move it into place on your server, however, it's important to organize your files. Also, you should have a good idea of what goes where to avoid lost files and broken links.

Questions to Ask Your Webmaster

The *webmaster* is the person who runs your web server. This person also might be your system administrator, help desk administrator, or network administrator. Before you can publish your site, you should get several facts from the webmaster about how the server is set up. The following list of questions will help you later in this book when you're ready to figure out what you can and cannot do with your server:

- **Where on the server will I put my files?** In most cases, someone will create a directory on the server where your files will reside. Know where that directory is and how to gain access to it.
- **What's the URL of my top-level directory?** This URL will usually be different from the actual path to your files.
- **What's the name of the system's default index file?** This file is loaded by default when a URL ends with a directory name. Usually it's `index.html` or `index.htm`, but it may be `default.htm`, or something else.
- **Can I run PHP, ASP, or other types of scripts?** Depending on your server, the answer to this question may be a flat-out "no," or you might be limited to certain programs and capabilities.
- **Do you support special plug-ins or file types?** If your site will include multimedia files (Flash, MP3, MP4 or others), your webmaster might need to configure the server to accommodate those file types. Make sure that the server properly handles special types of files before you create them.
- **Are there limitations on what or how much I can put up?** Some servers restrict pages to specific content (for example, only work-related pages) or restrict the amount of storage you can use. Make sure you understand these restrictions before you publish your content.
- **Is there a limit to the amount of bandwidth that my site can consume?** This is somewhat related to the previous question. Most web hosts allow you to transfer only a certain amount of data over their network over a given period of time before

they either cut you off or start charging you more money. You should ask what your bandwidth allotment is and make sure that you have enough to cover the traffic you anticipate. (The bandwidth allotment from most web hosts is more than enough for all but the most popular sites.)

- **Do you provide any canned scripts that I can use for my web pages?** If you aren't keen on writing your own scripts to add advanced features to your pages, ask your service provider whether it provides any scripts that might be of assistance. For example, many ISPs provide a script for creating an email contact form. Others might provide access to form-processing scripts, too.

Keeping Your Files Organized with Directories

Probably the easiest way to organize your site is to include all the files in a single directory. If you have many extra files—images, for example—you can put them in a subdirectory under that main directory. Your goal is to contain all your files in a single place rather than scatter them around. You can then set all the links in those files to be relative to that directory. This makes it easier to move the directory around to different servers without breaking the links.

Having a Default Index File and Correct Filenames

Web servers usually have a default index file that's loaded when a URL ends with a directory name rather than a filename. One of the questions you should ask your webmaster is, "What's the name of this default file?" For most web servers, this file is called `index.html`. Your home page, or top-level index, for each site should have this name so that the server knows which page to send as the default page. Each subdirectory should also have a default file if it contains any HTML files. If you use this default filename, the URL to that page will be shorter because you don't have to include the actual filename. For example, your URL might be http://www.example.com/pages/ rather than http://www.examplecom/pages/index.html.

CAUTION

If you don't put an index file in a directory, many web servers will enable people to browse the contents of the directory. If you don't want people to snoop around in your files, you should include an index file or use the web server's access controls to disable directory browsing.

Also, each file should have an appropriate extension indicating its type so the server can map it to the appropriate file type. If you've been reading this book in sequential order,

all your files should have this special extension already, and you shouldn't have any problems. Table 20.1 lists the common file extensions that you should be using for your files and multimedia.

TABLE 20.1 Common File Types and Extensions

Format	Extension
HTML	`.html`, `.htm`
ASCII Text	`.txt`
GIF	`.gif`
JPEG	`.jpg`, `.jpeg`
PNG	`.png`
Shockwave Flash	`.swf`
WAV Audio	`.wav`
MPEG Audio	`.mp3`
MPEG Video	`.mp4`
QuickTime Video	`.mov`
Portable Document Format	`.pdf`

If you're using multimedia files on your site that aren't part of this list, you might need to configure your server to handle that file type. You'll learn more about this issue later in this lesson.

Publishing Your Files

Got everything organized? Then all that's left is to move everything to the server. After your files have been uploaded to a directory that the server exposes on the Web, you're officially published on the Web. That's all there is to putting your pages online.

Where's the appropriate spot on the server, however? You should ask your webmaster for this information. Also, you should find out how to access that directory on the server, whether it's just copying files, using FTP to put them on the server, or using some other method.

Moving Files Between Systems

If you're using a web server that has been set up by someone else, usually you'll have to upload your web files from your system to theirs using FTP, SCP (secure copy), or some other method. Although the HTML markup within your files is completely cross-platform, moving the actual files from one type of system to another sometimes has its drawbacks. In particular, be careful to do the following:

- **Watch out for filename restrictions**—If your server is a PC and you've been writing your files on some other system, you might have to rename your files and the links to them to follow the correct file naming conventions. (Moving files you've created on a PC to some other system usually isn't a problem.)

  Also, watch out if you're moving files from a Macintosh to other systems. Make sure that your filenames don't have spaces or other funny characters in them. Keep your filenames as short as possible, use only letters and numbers, and you'll be fine.
- **Watch out for uppercase or lowercase sensitivity**—Filenames on computers running Microsoft Windows are not case-sensitive. On UNIX and Mac OS X systems, they are. If you develop your pages on a computer running Windows and publish them on a server that has case-sensitive filenames, you must make sure that you have entered the URLs in your links properly. If you're linking to a file named `About.html`, on your computer running Windows, `about.html` would work, but on a UNIX server it would not.
- **Be aware of carriage returns and line feeds**—Different systems use different methods for ending a line. The Macintosh uses carriage returns, UNIX uses line feeds, and DOS uses both. When you move files from one system to another, most of the time the end-of-line characters will be converted appropriately, but sometimes they won't. The characters that aren't converted can cause your file to come out double spaced or all on a single line when it's moved to another system.

  Most of the time, this failure to convert doesn't matter because browsers ignore spurious returns or line feeds in your HTML files. The existence or absence of either one isn't terribly important. However, it might be an issue in sections of text that you've marked up with `<pre>`; you might find that your well-formatted text that worked so well on one platform doesn't come out that way after it's been moved.

  If you do have end-of-line problems, you have two options. Many text editors enable you to save ASCII files in a format for another platform. If you know the platform to which you're moving, you can prepare your files for that platform before moving them.

Uploading Your Files

In the preceding list of tips about moving files, I mentioned FTP. FTP, short for File Transfer Protocol, is one of the ways to move files from your local computer to the server where they will be published, or to download them so that you can work on them, for that matter. Some other protocols that can be used to transfer files include SFTP (secure FTP), and SCP (secure copy). They all work a bit differently; the most important difference is that SCP and SFTP are encrypted, whereas FTP is not.

TIP

If your server provides multiple methods for uploading files, you should choose SCP or SFTP rather than FTP. With FTP, your password for the server will be transmitted unencrypted over the Internet. That's a security risk. It's preferable to use the encrypted uploading options.

A number of clients support FTP, SCP, and SFTP through the same interface. As long as you have the name of the server, your username, password, and the name of the directory where you want to put your files, you can use any of these clients to upload your web content.

One option that's often available is publishing files through your HTML editing tool. Many popular HTML and text editors have built-in support for FTP, SCP, and SFTP. You should definitely check your tool of choice to see whether it enables you to transfer files using FTP from directly within the application. Some popular tools that provide FTP support include Adobe Dreamweaver, Barebones BBEdit, and HTML-Kit. Text editors such as UltraEdit, Textmate, and jEdit support saving files to a server via FTP, too. If your HTML editor doesn't support FTP, or if you're transferring images, multimedia files, or even bunches of HTML files simultaneously, you'll probably want a dedicated FTP client. A list of some popular choices follows:

- **CuteFTP (Windows)**—http://www.globalscape.com/
- **FTP Explorer (Windows)**—http://www.ftpx.com/
- **FileZilla (Windows, OS X, Linux)**—http://filezilla-project.org/
- **Cyberduck (OS X)**—http://www.cyberduck.ch/
- **Transmit (OS X)**—http://www.panic.com/transmit/
- **Fetch (OS X)**—http://fetchsoftworks.com/

All the tools listed support FTP, SFTP, and SCP. How the FTP client is used varies depending on which client you choose, but there are some commonalities among all of them that you can count on (more or less). You'll start out by configuring a site consisting of the hostname of the server where you'll publish the files, your username and password, and perhaps some other settings that you can leave alone if you're just getting started.

CAUTION

> If you're sharing a computer with other people, you probably won't want to store the password for your account on the server in the FTP client. Make sure that the site is configured so that you have to enter your password every time you connect to the remote site.

After you've set up your FTP client to connect to your server, you can connect to the site. Depending on your FTP client, you should be able to simply drag files onto the window that shows the list of files on your site to upload them, or drag them from the listing on the server to your local computer to download them.

Troubleshooting

What happens if you upload all your files to the server and try to display your home page in your browser and something goes wrong? Here's the first place to look.

I Can't Access the Server

If your browser can't even get to your server, this probably isn't a problem you can fix. Make sure that you have entered the right server name and that it's a complete hostname (usually ending in `.com`, `.edu`, `.net`, or some other common suffix). Make sure that you haven't mistyped your URL and that you're using the right protocol. If your webmaster told you that your URL included a port number, make sure that you're including that port number in the URL after the hostname.

Also make sure that your network connection is working. Can you get to other URLs? Can you get to the top-level home page for the site itself?

If none of these ideas solve the problem, perhaps your server is down or not responding. Call your webmaster to find out whether she can help.

I Can't Access Files

What if all your files are showing up as Not Found or Forbidden? First, check your URL. If you're using a URL with a directory name at the end, try using an actual filename at the end. Double-check the path to your files; remember that the path in the URL might be different from the path on the actual disk. Also, keep case sensitivity in mind. If your file is `MyFile.html`, make sure that you're not trying `myfile.html` or `Myfile.html`.

If the URL appears to be correct, check the file permissions. On UNIX systems, all your directories should be world-executable, and all your files should be world-readable. You can ensure that all the permissions are correct by using the following commands:

```
chmod 755 filename
chmod 755 directoryname
```

TIP

Most FTP clients will allow you to modify file and directory permissions remotely.

I Can't Access Images

You can get to your HTML files just fine, but all your images are coming up as icons or broken icons? First, make sure that the references to your images are correct. If you've used relative pathnames, you shouldn't have this problem. If you've used full pathnames or file URLs, the references to your images may have been broken when you moved the files to the server. (I warned you[el].)

In some browsers, you get a pop-up menu when you select an image with the right mouse button. (Hold down the button on a Macintosh mouse.) Choose the View This Image menu item to try to load the image directly. This will give you the URL of the image where the browser thinks it's supposed to be (which might not be where *you* think it's supposed to be). You can often track down strange relative pathname problems this way.

If you're using Internet Explorer for Windows, you can also select the Properties option from the menu that appears when you right-click an image to see its address. You can check the address that appears in the Properties dialog box to see whether it points to the appropriate location.

If the references all look good and the images work just fine on your local system, the only other place a problem could have occurred is in transferring the files from one system to another.

My Links Don't Work

If your HTML and image files are working just fine but your links don't work, you most likely used pathnames for those links that applied only to your local system. For example, you might have used absolute pathnames or file URLs to refer to the files to which you're linking. As mentioned for images, if you used relative pathnames and avoided file URLs, you shouldn't have a problem.

My Files Are Being Displayed Incorrectly

Suppose you have an HTML file or a file in some multimedia format that's displayed correctly or links just fine on your local system. After you upload the file to the server and try to view it, the browser gives you gobbledygook. For example, it displays the HTML code itself instead of the HTML file, or it displays an image or multimedia file as text.

This problem can happen in two cases. The first is that you're not using the right file extensions for your files. Make sure that you're using one of the correct file extensions with the correct uppercase and lowercase.

The second case is that your server is not properly configured to handle your files. If all your HTML files have extensions of `.htm`, for example, your server might not understand that `.htm` is an HTML file. (Most modern servers do, but some older ones don't.) Or you might be using a newer form of media that your server doesn't understand. In either case, your server might be using some default content type for your files (usually `text/plain`), which your browser probably can't handle. This can happen with server-side scripts, too. If you put up `.php` files on a server that doesn't support PHP, the server will often send the scripts to the browser as plain text.

To fix this problem, you'll have to configure your server to handle the file extensions for the correct media. If you're working with someone else's server, you'll have to contact your webmaster and have him set up the server correctly. Your webmaster will need two types of information: the file extensions you're using and the content type you want him to return.

Registering and Advertising Your Web Pages

To get people to visit your website, you need to promote it. The more visible your site, the more hits it will attract.

A *hit* is a visit to your website. Be aware that although your site may get, say, 50 hits in a day, that doesn't necessarily mean that it was visited by 50 different people. It's just a record of the number of times a copy of your web page has been downloaded.

There are many ways to promote your site. You can make sure it's in search engine indexes, promote your site via social media, put the URL on your business cards, and so much more. The following sections describe each approach.

Getting Links from Other Sites

It doesn't take much surfing to figure out that the Web is huge. It seems like there's a site on every topic, and when it comes to popular topics, there may be hundreds or thousands of sites. After you've done the hard work of creating an interesting site, the next step is to get other people to link to it.

The direct approach often works best. Find other sites like your own and send a personal email to the people who run them introducing yourself and telling them that you have a site similar to theirs that they may be interested in. If they are, there's a good chance that they'll provide a link to your site. Oftentimes, there's a quid pro quo involved where you might link to someone else's site and ask them if they're interested in linking to yours in return.

This doesn't mean that you should go out and pester people or email them repeatedly if they don't do as you request. The subtle approach often works best. Figure out what kinds of people might be interested in what you're publishing, and let them know what you're up to. If you are launching a site for a new restaurant, it's worth searching for blogs that cover the city or neighborhood where the restaurant is and letting them know about the site. Many people are on the lookout for things to write about or link to, so if you have something of legitimate interest, they'll be glad to hear from you. Just make sure your email is to the point and that they know it was written to them personally, so they don't assume you're mass emailing people like them.

Promoting Your Site Through Social Media

First, what are social media? Most people define social media as websites that enable their users to socialize with one another. Sites such as Twitter, Facebook, and MySpace are popular examples. Weblogs can be considered social media, too. There are also link sharing sites like Digg and Reddit, where users can submit links, vote for them, or comment on them. Links that get more votes are featured more prominently on the site. Social media is about people connecting to one another, and promoting a site through social media is as simple as talking about your site on those sites. The tricky part is doing so in a way that makes you a valuable participant in the conversation rather than a tedious self-promoter.

In Lesson 22, "Content Management Systems and Publishing Platforms," I explain how you can integrate some of these social media sites with your own website, but first I explain ways you can use these sites to reach people who might be interested in your site. Many people talk about "viral" marketing. The concept is simple: Instead of purchasing an advertisement that may be displayed for hundreds or thousands of people, you tell just a few interested people about your site (or essay, or product, or movie, or whatever it is that you've created), and then they in turn share it with people they think

will be interested, and so on, until it has reached a large audience. The advantage, assuming that it works, is that it's inexpensive, and that your message has been delivered by people who the audience is actually willing to listen to—people they already know.The difficulty is in creating something that is interesting to large audiences in the first place, and in telling the right people about it so that they are interested in sharing in what you've created. Taking advantage of social media is one way to accomplish the second part of the task.

Regardless of the outlet, the two steps are to establish a presence and to be interesting. Twitter (http://twitter.com) is one of the most popular social media sites these days. After you've signed up for an account, you can follow other people on Twitter, and people who find you interesting will follow you. There are a lot of people on the Web giving advice on how to attract large numbers of followers, and there are a lot of people on Twitter who follow thousands of people in hopes that people will follow them in return.

Focusing on follower counts is the wrong approach. Remember, the goal with social media is to establish an audience of people who actually care about what you're doing. Let's say you've created a new website for knitting enthusiasts, and in hopes of promoting the site, you've created a Twitter account to go along with it.

NOTE

Creating a Twitter account is easy and free. To create a Twitter account, you need only supply an account name, a full name, an email address, and a password of your choosing. The account name and full name can be anything you like. After you've followed those steps, you're all set.

For starters, you should create posts on Twitter with links back to your site whenever you publish something new. You should also follow people who say interesting things, preferably on the subject of knitting. And you should respond to them when you have something interesting to say, too. If you do so, eventually they may follow you in return. If things go well, eventually you'll have a great outlet for promoting your site, and even if they go poorly, you'll be participating in a community of people who like to talk about the subject of your site—knitting. That's social media in a nutshell.

Creating a Facebook Page for Your Site

There's an additional way to promote your site on the popular social networking site Facebook. You can create a page that represents your site to the Facebook community. To do so, you need a Facebook account. After you've signed up, go to http://www.facebook.com/advertising/?pages and click Create a Page. You'll be taken to the Create a Page form, shown in Figure 20.1.

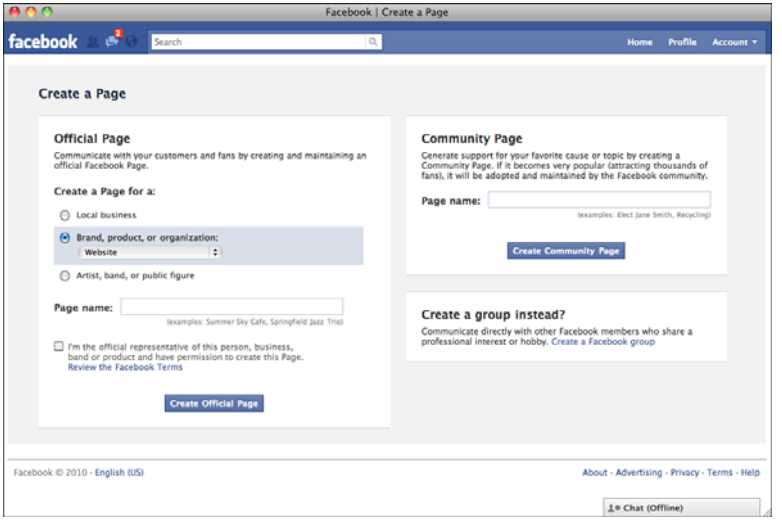


FIGURE 20.1
The Facebook Create a Page form.

In the form, I've already selected Brand, Product, or Organization and then chosen Website from the select list. At that point, I just have to enter a name for my page to create it. The brand new page I created for this book appears in Figure 20.2.

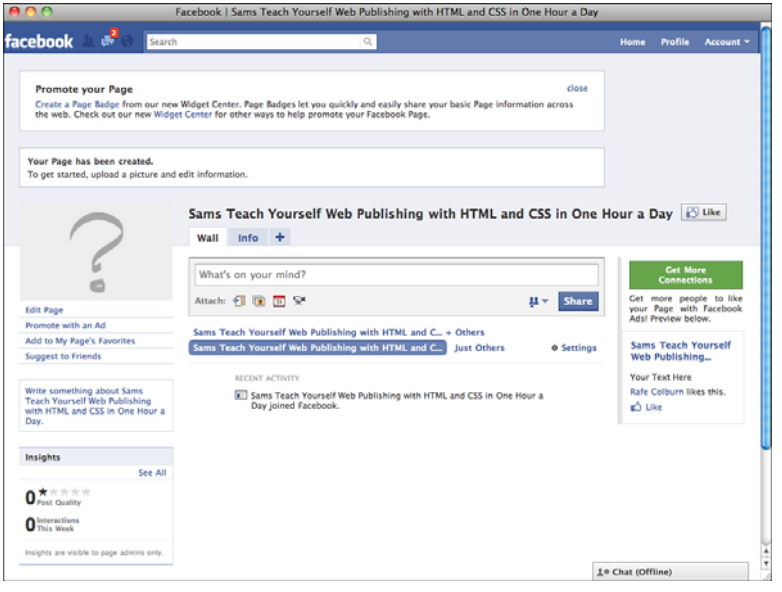


FIGURE 20.2
The new Facebook page I created for this book.

After the page has been created, I can customize it in a variety of ways, controlling who's allowed to post on it and what kinds of content they're allowed to post. The Facebook page gives Facebook users who are interested in this book a place to congregate to discuss it, share links related to the book, and meet one another.

Site Indexes and Search Engines

Nearly all web users know how to find things using search engines, and you'll want to make sure that they can find your site. Search engines work by creating an index of all the sites they can find. You need to be sure to add your site to the index when you publish it so that search engines will start including it in search results. As long as people link to your site, search engines will find it eventually whether you tell them about it or not, but asking them to index your site will ensure that it's added immediately. Here's a list of the top four search engines: :

Google	http://www.google.com
Bing	http://www.bing.com
Yahoo	http://www.yahoo.com
Ask Jeeves	http://www.ask.com

I'm going to describe how to submit your sites to some of the popular search engines. There's a set of interlocking relationships among search engine providers that can make it difficult to keep track of who is providing search functionality for whom. The search engines listed previously maintain their own indexes. After your site is included in their index, it will be available via all the search engines that use their index, too.

NOTE

The search engine descriptions are brief, and because of the rapidly changing nature of the search engine industry, may be out of date by the time you read them. For more information about search engines, I strongly recommend Search Engine Watch at http://www.searchenginewatch.com.

Google

Google is currently the most popular search engine. Its search results are ranked based not only on how frequently the search terms appear on a listed page, but also on the number of other pages in the index that link to that page. So, a popular page with thousands of incoming links will be ranked higher than a page that has only a few incoming links.

This search algorithm does a remarkably good job of pushing the most relevant sites to the top of the search results. It also rewards people who publish useful, popular sites rather than those who've figured out how to manipulate the algorithms that other search engines use. Many sites that aren't dedicated to providing search functionality use Google's index, so getting into the Google index provides wide exposure. As of summer 2010, Google's share of the search engine market is 65%.

How Search Results Are Ranked

Every search engine has an algorithm that ranks sites based on their relevance. It might take into account how many times the keyword you enter appears on the page, whether it appears inside heading tags or in the page title, or whether it appears in text inside links. It might also take into account how high on the page your search terms appear. Such algorithms are trade secrets within the search engine industry, but some of them have been unraveled to greater or lesser degrees. Armed with this information, some site authors write their pages in such a way that search engines will give them a higher relevance ranking than they deserve. For example, some sites have long titles with lots of information in hopes of appearing first in search results.

Yahoo!

Yahoo! has been around since 1994. It provides both a human-edited directory of the Web, which I discussed earlier, and web search. Yahoo's search engine currently uses its own index, but it has an agreement with Microsoft to use the Bing index at some point in the future. Yahoo!'s share of the search engine market is around 17% at the time of this writing.

Microsoft Bing

Bing is Microsoft's web search offering. Like Google, Microsoft maintains its own index of the Web. Bing was launched in May 2009, and as of May 2010 had an 11% market share among search engines.

Ask.com

Ask.com is another search engine that maintains its own index. Ask.com indexes fewer pages than Google or Yahoo! Ask.com controls only about 2% of the search engine market as of summer 2010, but is listed because, like the other sites listed previously, it has its own index.

Search Engine Optimization

Not only do you want to make sure your site is included in the popular search engines, but you also want to make sure that it shows up near the top of the results when people are searching for topics that are related to your site. If you create a site about model railroads, you want your site to appear as high as possible in the results for searches like "model railroad" and "model train."

Unfortunately, search engines don't publish instructions on how to make your site rank near the top of their index. Before Google, search engines ranked sites mostly on the basis of their content. The more prominently a term was placed on your website, the higher it would be ranked for that term. So if your model trains page had "model trains" in a page title, or in heading tags on the page, that page would be more highly ranked for the terms "model train." That's why you sometimes see pages with lots and lots of words listed in the title—it's an attempt to improve search engine rankings for those words.

At one point, search engines enabled you to provide hints about your site by way of `meta` tags. There was a tag that enabled you to specify a description for the page, and another that enabled you to list keywords associated with each page. The content of the tags was invisible; it was only used to help search engines with indexing. Unfortunately, people who published websites immediately started abusing `meta` tags, putting in keywords that were not related to the page content, or putting in far too many keywords to try to gain a higher ranking. Now all the popular search engines disregard `meta` tags entirely.

Search engines now consider not only what's on your site but also who links to it in determining the relevance of your pages, so the more sites that link to yours, the better your placement will be in search engine results. A number of companies sell search engine optimization services that attempt to exploit this method of improving search rankings by paying popular sites to link to your site, or even creating fake sites and filling them with links to their clients. You should avoid these types of services and be wary of search engine optimization services in general.

Instead of worrying too much about how to make your site more friendly to search engines, you should worry about writing good HTML and making your site friendly for users. As your site gains in popularity, your search engine ranking will come along. One thing that can help is writing good, descriptive titles, and making sure to use heading tags for headings rather than just using large fonts.

Paying for Search Placement

All the popular search engines have programs that allow you to pay for search placement. In other words, you can agree to pay to have a link to your site displayed when users enter search terms that you choose. Generally this service is priced on a per-click basis—you pay every time a user clicks on the link, up to a maximum that you set. After you've used up your budget, your advertisement doesn't appear any more.

Most search engines display paid links separately from the regular search results, but this approach still provides a way to get your site in front of users who may be interested immediately. You just have to be willing to pay.

Business Cards, Letterhead, Brochures, and Advertisements

Although the Internet is a wonderful place to promote your new website, many people fail to consider some other great advertising methods.

Most businesses spend a considerable amount of money each year producing business cards, letterhead, and other promotional material. These days it's rare to see any of these materials without web and email information on them. By printing your email address and home page URL on all your correspondence and promotional material, you can reach an entirely new group of potential visitors.

Even your email signature is a good place to promote your site. Just put in a link and the title or a short description, so that everyone you correspond with can see what you're publishing on the Web.

When you're promoting your website, the bottom line is lateral thinking. You need to use every tool at your disposal if you want to have a successful and active site.

Finding Out Who's Viewing Your Web Pages

Now you've got your site up on the Web and ready to be viewed, you've advertised and publicized it to the world, and people are flocking to it in droves. Or are they? How can you tell? You can find out in a number of ways, including using log files and access counters.

Log Files

The best way to figure out how often your pages are being seen, and by whom, is to get access to your server's log files. How long these log files are kept depends on how your server is configured. The logs can take up a lot of disk space, so some hosting providers remove old logs frequently. If you run your own server, you can keep them as long as you like, or at least until you run out of room. Many commercial web providers allow you to view your own web logs or get statistics about how many visitors are accessing your pages and from where. Ask your webmaster for help.

If you do get access to these raw log files, you'll most likely see a whole lot of lines that look something like the following. (I've broken this one up into two lines so that it fits on the page.)

```
vide-gate.coventry.ac.uk - - [17/Feb/2003:12:36:51 -0700]
  "GET /index.html HTTP/1.0" 200 8916
```

What does this information mean? The first part of the line is the site that accessed the file. (In this case, it was a site from the United Kingdom.) The two dashes are used for authentication. (If you have login names and passwords set up, the username of the person who logged in and the group that person belonged to will appear here.) The date and time the page was accessed appear inside the brackets. The next part is the actual filename that was accessed; here it's the `index.html` at the top level of the server. The `GET` part is the actual HTTP command the browser used; you usually see `GET` here. Finally, the last two numbers are the HTTP status code and the number of bytes transferred. The status code can be one of many things: `200` means the file was found and transferred correctly; `404` means the file was not found. (Yes, it's the same status code you get in error pages in your browser.) Finally, the number of bytes transferred usually will be the same number of bytes in your actual file; if it's a smaller number, the visitor interrupted the load in the middle.

Most web hosts provide log processing software that will take the logs generated by the server when users visit your site and turn them into reports, often with graphs and other visual aids, that you can use to easily see how many users are visiting your site as well as how those servers are finding your site, whether it's through search engines or links on other web pages. You'll want to check out the support site for your web host to determine how to set things up so that your logs are processed and find out the URL of the reports that are generated.

Google Analytics

There are other ways to keep track of who's visiting your site and what pages they're viewing, too. Processing log files is one way to get an idea of who's visiting your site. Another option is to use Google Analytics, a tool provided for free by Google that keeps track of all the visitors to your site and generates reports about your visitors.

The nice thing about Google Analytics is that you don't have to deal with any log files. Google Analytics works by providing you with a code that uniquely identifies your site. On each of the pages that you want to track, you include a reference to a JavaScript file that Google provides, and pass in the code for your site. Whenever users visit the pages with a link to the tracking script, Google records information about their visit. Google then uses this information to create the reports for you.

One particularly nice thing about Google Analytics is that you can usually add it to your site even if the site is on a server you don't control. So if you create a weblog (blog) on a site like Tumblr, you can edit the theme of the site and paste in the Google Analytics code.

Installing Google Analytics

To get started with installation, you'll need to go to the Google Analytics website, http://www.google.com/analytics/ and sign up for an Analytics account. If you don't already have a Googe account, you'll need to sign up for one, too.

After you sign up for your account, you need to create a profile for your website. Click the Add Website Profile link on the Google Analytics home page to create the profile, and you'll see the form in Figure 20.3.

To add a profile, you just enter the URL of your website in the form provided and choose the time zone for your site. After you've saved your new profile, Google provides the code to paste into your own web page so that customer visits can be tracked. The code itself is a snippet of JavaScript that loads the Google tracking code. To install the Google tracking code on your site, copy the code that Google provides into your own pages. Google recommends that you paste the tag just inside the closing `<head>` tag on your pages, but for reasons of performance, it probably makes more sense to paste it just before the closing `<body>` tag. To start out, edit the HTML for your site's home page and paste in the Google Analytics code. Upload the page to your server if necessary, and then visit that page in your browser.

After the page has been loaded with the Google Analytics tracking code in place, Google Analytics will indicate that it has started tracking visits to your site. At that point, add the Google tracking code to your other pages and upload them, too.

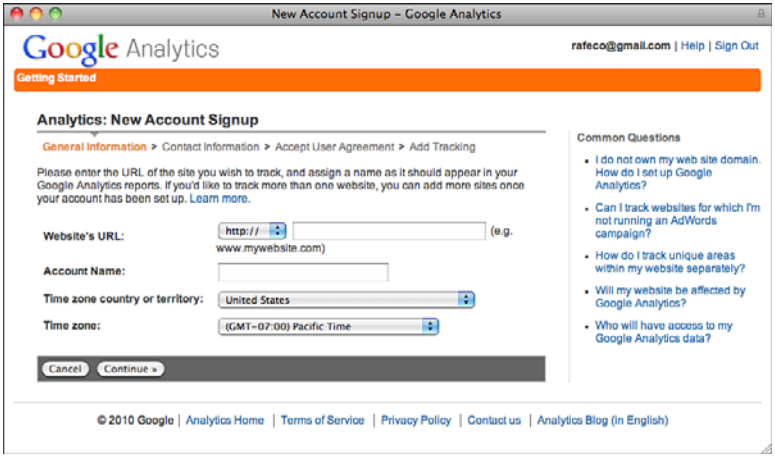


FIGURE 20.3
Setting up a new Google Analytics profile.

Using the Google Analytics Reports

After Google Analytics has been installed, it will start creating reports for your site about 24 hours later. To view the main report for your site, just click View Report for your site in the Website Profiles list. The Dashboard shows some basic statistics about use of your site—how many visits you've gotten each day for the past month, a map showing where most of your visitors come from, and which pages on your site are the most popular. You can see an example of the Dashboard in Figure 20.4.

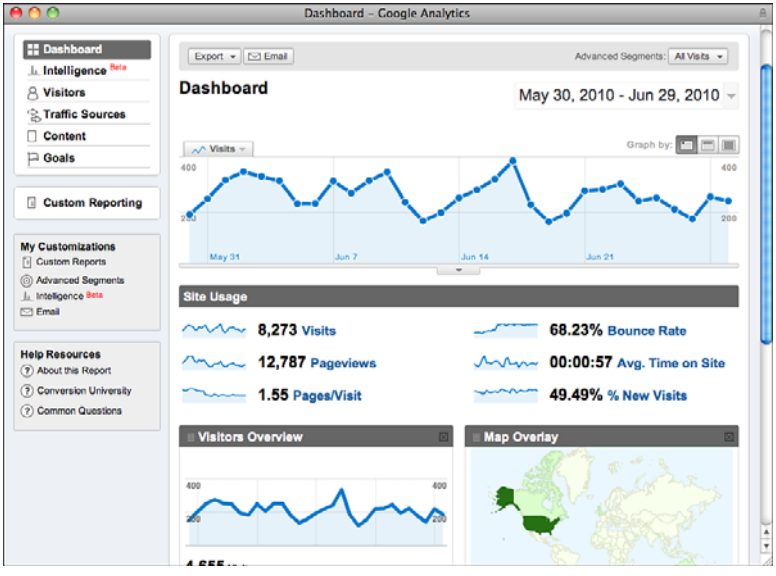


FIGURE 20.4
The Google Analytics Dashboard.

Other numbers on the page provide insight into how users are interacting with your site. Bounce Rate shows the percentage of users who leave after visiting your landing page instead of sticking around to visit more pages on your site. The average pages per visit and average time on site provide a further idea of the degree to which users are drilling down on your site. In some cases, low numbers here may be fine. If your page is a set of links to other sites, a high bounce rate and low time on the site may indicate that users are finding what they're looking for and following the links. Your interpretation of the statistics should be based on your goals.

Each of the reports on the Dashboard links to a report with more detailed information. For example, if you click the report link for Traffic Sources, you'll see a more detailed breakdown of where your traffic originated, including which search terms people used to find your site.

One report shows which browsers and operating systems your visitors are using, so that you can figure out which features your audiences will be able to take advantage of. Other reports show how many of your users visited for the first time and how many are repeat visitors. There are reports that show which sites link to yours. Keeping a close eye on your Analytics reports will enable you to figure out which parts of your site are working and which aren't, whether you use Google Analytics or some other analytics package.

Summary

In this lesson, you published your site on the Web through the use of a web server, either one installed by you or that of a network provider. You learned what a web server does and how to get one, how to organize your files and install them on the server, and how to find your URL and use it to test your pages. You also learned the many ways that you can advertise and promote your site, and how to use log files and Google Analytics to keep track of the number of visitors. At last, you're on the Web and people are coming to visit!

Workshop

As always, we wrap up the lesson with a few questions, quizzes, and exercises. Here are some pointers and refreshers on how to promote your website.

Q&A

Q I've published my pages at an ISP I really like. The URL is something like http://www.thebestisp.com/users/mypages/. Instead of this URL, I'd like to have my own hostname, something like http://www.mypages.com/. How can I do this?

A You have two choices. The easiest way is to ask your ISP whether you're allowed to have your own domain name. Many ISPs have a method for setting up your domain so that you can still use their services and work with them—it's only your URL that changes. Note that having your own hostname might cost more money, but it's the way to go if you really must have that URL. Many web hosting services have plans starting as low as $5 a month for this type of service, and it currently costs as little as $16 to register your domain for two years.

The other option is to set up your own server with your own domain name. This option could be significantly more expensive than working with an ISP, and it requires at least some background in basic network administration.

Q There are so many search engines! Do I have to add my URL to all of them?

A No, mainly because eventually they will find your site whether you add it to them or not. Adding your URL to a search engine may get it into the results more quickly, so if you already know about a search engine and can submit your site, do so. Otherwise, don't worry about it.

Quiz

1. What's the basic function of a web server?
2. What are default index files, and what's the advantage of using them in all directories?
3. What are some things that you should check immediately after you upload your web pages?
4. Name some of the ways that you can promote your website.
5. What's a hit?

Quiz Answers

1. A web server is a program that sits on a machine connected to the Internet (or an intranet). It determines which resource is associated with a URL and delivers that resource to the user.

2. The default index file is loaded when a URL ends with a directory name rather than a filename. Typical examples of default index files are `index.html`, `index.htm`, and `default.htm`. If you use default filenames, you can use a URL such as http://www.mysite.com/ rather than http://www.mysite.com/index.html to get to the home page in the directory.
3. Make sure that your browser can reach your web pages on the server, that you can access the files on your website, and that your links and images work as expected. After you've determined that everything appears the way you think it should, have your friends and family test your pages in other browsers.
4. Some ways you can promote your site include major web directories and search engines, listings on business cards and other promotional materials, and web rings.
5. A hit is a request for any file from your website.

Exercises

1. Start shopping around and consider where you want to host your website. Find a couple of web hosting firms that look like good options and do some research online to see what their existing customers have to say about them.
2. Upload and test a practice page to learn the process, even if it's just a blank page that you'll add content to later. You might work out a few kinks this way before you actually upload all your hard work on the Web.
3. Visit some of the search engines listed in this lesson to obtain a list of the sites where you want to promote your web page. Review each of the choices to see whether there are special requirements for listing your page.
4. Sign up for a Google Analytics account and install it on your site. Explore the reports to see what kind of information it provides about your site.

LESSON 21
Taking Advantage of the Server

At this point, you've learned how to publish websites using Hypertext Markup Language (HTML). This lesson takes things a step further and explains how to build dynamic websites using scripts on the server. Most websites utilize some kind of server-side processing. Search engines take the user's request and search an index of web pages on the server. Online stores use server-side processing to look up items in the inventory, keep track of the user's shopping cart, and handle the checkout process. Newspaper websites keep articles in a database and use server-side processing to generate the article pages. This lesson introduces server-side programming using the PHP language. PHP is the most common scripting platform provided by web hosts, can be easily installed on your own computer, and is completely free. It's also easy to get started with. Even if you wind up developing your applications using some other scripting language, you can apply the principles you'll learn in this lesson to those languages.

In this lesson, you'll learn the following:

- How PHP works
- How to set up a PHP development environment
- The basics of the PHP language
- How to process form input
- Using PHP includes

How PHP Works

PHP enables programmers to include PHP code in their HTML documents, which is processed on the server before the HTML is sent to the browser. Normally, when a user submits a request to the server for a web page, the server reads the HTML file and sends its contents back in response. If the request is for a PHP file and the server supports PHP, the server looks for PHP code in the document, executes it, and includes the output of that code in the page in place of the PHP code. Here's a simple example:

```
<!DOCTYPE html>
<html>
<head><title>A PHP Page</title></head>
<body>
<?php echo "Hello world!"; ?>
</body>
</html>
```

If this page is requested from a web server that supports PHP, the HTML sent to the browser will look like this:

```
<!DOCTYPE html>
<html>
<head><title>A PHP Page</title></head>
<body>
Hello world!
</body>
</html>
```

When the user requests the page, the web server determines that it is a PHP page rather than a regular HTML page. If a web server supports PHP, it usually treats any files with the extension `.php` as PHP pages. Assuming this page is called something like `hello.php`, when the web server receives the request, it scans the page looking for PHP code and then runs any code it finds. PHP code is distinguished from the rest of a page by PHP tags, which look like this:

```
<?php your code here ?>
```

Whenever the server finds those tags, it treats whatever is within them as PHP code. That's not so different from the way things work with JavaScript, where anything inside `<script>` tags is treated as JavaScript code.

In the example, the PHP code contains a call to the `echo` function. This function prints out the value of whatever is passed to it. In this case, I passed the text "Hello world!" to the function, so that text is included in the page. The concept of functions should also be familiar to you from the lesson on JavaScript. Just like JavaScript, PHP lets you define your own functions or use functions built in to the language. `echo` is a built-in function.

Statements in PHP, as in JavaScript, are terminated with a semicolon. (You can see the semicolon at the end of the statement in the example.) There's no reason why you can't include multiple statements within one PHP tag, like this:

```
<?php
  echo "Hello ";
  echo "world!";
?>
```

PHP also provides a shortcut if all you want to do is print the value of something to a page. Instead of using the full PHP tag, you can use the expression tag, which just echoes a value to the page. Instead of using

```
<?php echo "Hello world!"; ?>
```

You can use this:

```
<?= "Hello world!" ?>
```

Replacing `php` with `=` enables you to leave out the call to the `echo` function and the semicolon. This style of tag is referred to as a *short tag*. Not all PHP installations have short tags enabled.

Getting PHP to Run on Your Computer

Before you can start writing your own PHP scripts, you need to set up a PHP environment. The easiest approach is probably to sign up for a web hosting account that provides PHP support. Even if you do so, though, there are some advantages to getting PHP to work on your own computer. You can edit files with your favorite editor and then test them right on your own computer rather than uploading them to see how they work. You'll also be able to work on them even if you're not online. Finally, you can keep from putting files on a server that your users see without your having tested them first.

To process PHP pages, you need the PHP interpreter and a web server that works with the PHP interpreter. The good news is that PHP and the most popular web server, Apache, are both free, open source software. The bad news is that getting PHP up and running can be a bit of a technical challenge.

Fortunately, if you're a Windows or Mac user, someone else has done this hard work for you. A tool called XAMPP, available for both Windows and OS X, bundles up versions of Apache, PHP, and MySQL (a database useful for storing data associated with web applications) that are already set up to work together. (The last P is for Perl, another scripting language.) You can download it from http://www.apachefriends.org/en/xampp.html.

If you're a Mac user, you also have the option of using MAMP, another free package that combines Apache, PHP, and MySQL. It can be downloaded from http://www.mamp.info.

Mac users also have the option of using the version of Apache and PHP that are included with OS X.

After you've installed XAMPP (or MAMP), you just have to start the application to get a web server up and running that you can use to develop your pages. To test your PHP pages, you can put them in the `htdocs` directory inside the XAMPP install directory. For example, if you want to test the `hello.php` page I talked about earlier, you could put it in the `htdocs` directory. To view it, just go to http://localhost/hello.php.

If that doesn't work, make sure that XAMPP has started the Apache server. If you're using MAMP, the steps are basically the same. Just put your pages in the `htdocs` folder, as with XAMPP.

The PHP Language

When you think about the English language, you think about it in terms of parts of speech. Nouns name things, verbs explain what things do, adjectives describe things, and so on. Programming languages are similar. A programming language is made up of various "parts of speech," too. In this section, I explain the parts of speech that make up the PHP language—comments, variables, conditional statements, and functions.

It might be helpful to think back to the lesson on JavaScript as you read this lesson. PHP and JavaScript share a common ancestry, and many of the basic language features are similar between the two. If things such as the comment format, curly braces, and control statements look similar from one to the other, it's because they are.

Comments

Like HTML and JavaScript, PHP supports comments. PHP provides two comment styles: one for single-line comments, and another for multiple comments. (If you're familiar with comments in the C or Java programming language, you'll notice that PHP's are the same.) First, single-line comments. To start a single-line comment, use `//` or `#`. Everything that follows either on a line is treated as a comment. Here are some examples:

```
// My function starts here.
$color = 'red'; // Set the color for text on the page
# $color = 'blue';
$color = $old_color; # Sets the color to the old color.
// $color = 'red';
```

The text that precedes `//` is processed by PHP, so the second line assigns the `$color` variable. On the third line, I've turned off the assignment by commenting it out. PHP also supports multiple-line comments, which begin with `/*` and end with `*/`. If you want to comment out several lines of code, you can do so like this:

```
/*
$color = 'red';
$count = 55; // Set the number of items on a page.
// $count = $count + 1;
*/
```

PHP ignores all the lines inside the comments. Note that you can put the `//` style comment inside the multiline comment with no ill effects. You cannot, however, nest multiline comments. This is illegal:

```
/*
$color = 'red';
$count = 55; // Set the number of items on a page.
/* $count = $count + 1; */
*/
```

NOTE The generally accepted style for PHP code is to use `//` for single-line comments rather than `#`.

Variables

Variables just provide a way for the programmers to assign a name to a piece of data. In PHP, these names are preceded by a dollar sign (`$`). Therefore, you might store a color in a variable called `$color` or a date in a variable named `$last_published_at`. Here's how you assign values to those variables:

```
$color = "red";
$last_published_at = time();
```

The first line assigns the value `"red"` to `$color`; the second returns the value returned by the built-in PHP function `time()` to `$last_published_at`. That function returns a timestamp represented as the number of seconds since the "UNIX epoch."

One thing you should notice here is that you don't have to indicate what kind of item you'll be storing in a variable when you declare it. You can put a string in it, as I did when I assigned `"red"` to `$color`. You can put a number in it, as I did with `$last_published_at`. I know that the number is a timestamp, but as far as PHP is

concerned, it's just a number. What if I want a date that's formatted to be displayed rather than stored in seconds so that it can be used in calculations? I can use the PHP `date()` function. Here's an example:

```
$last_published_at = date("F j, Y, g:i a");
```

This code formats the current date so that it looks something like "June 10, 2010, 8:47 pm." As you can see, I can change what kind of information is stored in a variable without doing anything special. It just works. The only catch is that you have to keep track of what sort of thing you've stored in a variable when you use it. For more information about how PHP deals with variable types, see http://www.php.net/manual/en/language.types.type-juggling.php.

Despite the fact that variables don't have to be declared as being associated with a particular type, PHP does support various data types, including string, integer, and float (for numbers with decimal points). Not all variable types work in all contexts. One data type that requires additional explanation is the array data type.

Arrays

The variables you've seen so far in this lesson have all been used to store single values. Arrays are data structures that can store multiple values. You can think of them as lists of values, and those values can be strings, numbers, or even other arrays. To declare an array, use the built-in `array` function:

```
$colors = array('red', 'green', 'blue');
```

This declaration creates an array with three elements in it. Each element in an array is numbered, and that number is referred to as the *index*. For historical reasons, array indexes start at 0, so for the preceding array, the index of `red` is 0, the index of `green` is 1, and the index of `blue` is 2. You can reference an element of an array using its index, like this:

```
$color = $colors[1];
```

By the same token, you can assign values to specific elements of an array, too, like this:

```
$colors[2] = 'purple';
```

You can also use this method to grow an array, as follows:

```
$colors[3] = 'orange';
```

What happens if you skip a few elements when you assign an item to an array, as in the following line?

```
$colors[8] = 'white';
```

In this case, not only will element 8 be created, but elements 4 through 7 will be created, too. If you want to add an element onto the end of an array, you just leave out the index when you make the assignment, like this:

```
$colors[] = 'yellow';
```

In addition to arrays with numeric indexes, PHP supports associative arrays, which have indexes supplied by the programmer. These are sometimes referred to as *dictionaries* or as *hashes*. Here's an example that shows how they are declared:

```
$state_capitals = array(
  'Texas' => 'Austin',
  'Louisiana' => 'Baton Rouge',
  'North Carolina' => 'Raleigh',
  'South Dakota' => 'Pierre'
);
```

When you reference an associative array, you do so using the keys you supplied, as follows:

```
$capital_of_texas = $state_capitals['Texas'];
```

To add a new element to an associative array, you just supply the new key and value, like this:

```
$state_capitals['Pennsylvania'] = 'Harrisburg';
```

If you need to remove an element from an array, just use the built-in `unset()` function, like this:

```
unset($colors[1]);
```

The element with the index specified will be removed, and the array will decrease in size by one element, too. The indexes of the elements with larger indexes than the one that was removed will be reduced by one. You can also use `unset()` to remove elements from associative arrays, like this:

```
unset($state_capitals['Texas']);
```

Array indexes can be specified using variables. You just put the variable reference inside the square brackets, like this:

```
$i = 1;
$var = $my_array[$i];
```


This also works with associative arrays:

```
$str = 'dog';
$my_pet = $pets[$str];
```

As you'll see a bit further on, the ability to specify array indexes using variables is a staple of some kinds of loops in PHP.

As you've seen, nothing distinguishes between a variable that's an array and a variable that holds a string or a number. PHP has a built-in function named `is_array()` that returns true if its argument is an array and false if the argument is anything else. Here's an example:

```
is_array(array(1, 2, 3));  // returns true
is_array('tree'); // returns false
```

To determine whether a particular index is used in an array, you can use PHP's `array_key_exists()` function. This function is often used to do a bit of checking before referring to a particular index, for example:

```
if (array_key_exists($state_capitals, "Michigan")) {
    echo $state_capitals["Michigan"];
}
```

As mentioned previously, it's perfectly acceptable to use arrays as the values in an array. Therefore, the following is a valid array declaration:

```
$stuff = ('colors' => array('red', 'green', 'blue'),
          'numbers' => array('one', 'two', 'three'));
```

In this case, I have an associative array that has two elements. The values for each of the elements are arrays themselves. I can access this data structure by stacking the references to the array indexes, like this:

```
$colors = $stuff['colors']; // Returns the list of colors.
$color = $stuff['colors'][1]; // Returns 'green'
$number = $stuff['numbers'][0]; // Returns 'one'
```

Strings

The most common data type you'll work with in PHP is the string type. A string is just a series of characters. An entire web page is a string, as is a single letter. To define a string, just place the characters in the string within quotation marks. Here are some examples of strings:

```
"one"
"1"
"I like publishing Web pages."
"This string
spans multiple lines."
```

Take a look at the last string in the list. The opening quotation mark is on the first line, and the closing quotation mark is on the second line. In PHP, this is completely valid. In some programming languages, strings that span multiple lines are illegal—not so in PHP, where strings can span as many lines as you like, so long as you don't accidentally close the quotation marks.

There's more to strings than just defining them. You can use the `.` operator to join strings, like this:

```
$html_paragraph = "<p>" . $paragraph . "</p>";
```

The `$html_paragraph` variable will contain the contents of `$paragraph` surrounded by the opening and closing paragraph tag. The `.` operator is generally referred to as the *string concatenation operator*.

Up to this point, you might have noticed that sometimes I've enclosed strings in double quotation marks, and that other times I've used single quotation marks. They both work for defining strings, but there's a difference between the two. When you use double quotation marks, PHP scans the contents of the string for variable substitutions and for special characters. When you use single quotation marks, PHP just uses whatever is in the string without checking to see whether it needs to process the contents.

Special characters are introduced with a backslash, and they are a substitute for characters that might otherwise be hard to include in a string. For example, `\n` is the substitute for a newline, and `\r` is the substitute for a carriage return. If you want to include a newline in a string and keep it all on one line, just write it like this:

```
$multiline_string = "Line one\nLine two";
```

Here's what I mean by variable substitutions. In a double-quoted string, I can include a reference to a variable inside the string, and PHP will replace it with the contents of the variable when the string is printed, assigned to another variable, or otherwise used. In other words, I could have written the preceding string-joining example as follows:

```
$html_paragraph = "<p>$paragraph</p>";
```

PHP will find the reference to `$paragraph` within the string and substitute its contents. On the other hand, the literal value "$paragraph" would be included in the string if I wrote that line like this:

```
$html_paragraph = '<p>$paragraph</p>';
```

You need to do a bit of extra work to include array values in a string. For example, this won't work:

```
$html_paragraph = "<p>$paragraph['intro']</p>";
```

You can include the array value using string concatenation:

```
$html_paragraph = "<p>" . $paragraph['intro'] . "</p>";
```

You can also use array references within strings if you enclose them within curly braces, like this:

```
$html_paragraph = "<p>{$paragraph['intro']}</p>";
```

One final note on defining strings is escaping. As you know, quotation marks are commonly used in HTML as well as in PHP, especially when it comes to defining attributes in tags. There are two ways to use quotation marks within strings in PHP. The first is to use the opposite quotation marks to define the string that you're using within another string. Here's an example:

```
$tag = '<p class="important">';
```

I can use the double quotes within the string because I defined it using single quotes. This particular definition won't work, though, if I want to specify the class using a variable. If that's the case, I have two other options:

```
$tag = "<p class=\"$class\">";
$tag = '<p class="' . $class . '">';
```

In the first option, I use the backslash character to "escape" the double quotes that occur within the string. The backslash indicates that the character that follows is part of the string and does not terminate it. The other option is to use single quotes and use the string concatenation operator to include the value of `$class` in the string.

Conditional Statements

Conditional statements and loops are the bones of any programming language. PHP is no different. The basic conditional statement in PHP is the `if` statement. Here's how it works:

```
if ($var == 0) {
  echo "Variable set to 0.";
}
```

The code inside the brackets will be executed if the expression in the `if` statement is true. In this case, if `$var` is set to anything other than 0, the code inside the brackets will not be executed. PHP also supports `else` blocks, which are executed if the expression in the `if` statement is false. They look like this:

```
if ($var == 0) {
  echo "Variable set to 0.";
} else {
```

```
  echo "Variable set to something other than 0.";
}
```

When you add an `else` block to a conditional statement, it means that the statement will always do something. If the expression is true, it will run the code in the `if` portion of the statement. If the expression is not true, it will run the code in the `else` portion. Finally, there's `elseif`:

```
if ($var == 0) {
  echo "Variable set to 0.";
} elseif ($var == 1) {
  echo "Variable set to 1.";
} elseif ($var == 2) {
  echo "Variable set to 2.";
} else {
  echo "Variable set to something other than 0, 1, or 2.";
}
```

As you can see, `elseif` allows you to add more conditions to an `if` statement. In this case, I added two `elseif` conditions. There's no limit on `elseif` conditions—you can use as many as you need. Ultimately, `elseif` and `else` are both conveniences that enable you to write less code to handle conditional tasks.

PHP Conditional Operators

It's hard to write conditional statements if you don't know how to write a Boolean expression. First of all, Boolean means that an expression (which you can think of as a statement of fact) is either true or false. Here are some examples:

```
1 == 2 // false
'cat' == 'dog' // false
5.5 == 5.5 // true
5 > 0 // true
5 >= 5 // true
5 < 10 // true
```

PHP also supports logical operators, such as "not" (which is represented by an exclamation point), "and" (`&&`), and "or" (`||`). You can use them to create expressions that are made up of multiple individual expressions, like these:

```
1 == 1 && 2 == 4 // false
'blue' == 'green' || 'blue' == 'red' // false
!(1 == 2) // true, because the ! implies "not"
!(1 == 1 || 1 == 2) // false, because ! negates the expression inside the ()
```

Furthermore, individual values also evaluate to true or false on their own. Any variable set to anything other than 0 or an empty string (`""` or `''`) will evaluate as true, including an array with no elements in it. So if `$var` is set to 1, the following condition will evaluate as true:

```
if ($var) {
  echo "True.";
}
```

If you want to test whether an array is empty, use the built-in function `empty()`. So if `$var` is an empty array, `empty($var)` will return `true`. Here's an example:

```
if (empty($var)) {
  echo "The array is empty.";
}
```

You can find a full list of PHP operators at http://www.php.net/manual/en/language.operators.php.

Loops

PHP supports several types of loops, some of which are generally more commonly used than others. As you know from the JavaScript lesson, loops execute code repeatedly until a condition of some kind is satisfied. PHP supports several types of loops: `do...while`, `while`, `for`, and `foreach`. I discuss them in reverse order.

`foreach` Loops

The `foreach` loop was created for one purpose—to enable you to process all the elements in an array quickly and easily. The body of the loop is executed once for each item in an array, which is made available to the body of the loop as a variable specified in the loop statement. Here's how it works:

```
$colors = array('red', 'green', 'blue');
foreach ($colors as $color) {
  echo $color . "\n";
}
```

This loop prints each of the elements in the `$colors` array with a linefeed after each color. The important part of the example is the `foreach` statement. It specifies that the array to iterate over is `$colors`, and that each element should be copied to the variable `$color` so that it can be accessed in the body of the loop.

The `foreach` loop can also process both the keys and values in an associative array if you use slightly different syntax. Here's an example:

```
$synonyms = array('large' => 'big',
                  'loud' => 'noisy',
                  'fast' => 'rapid');

foreach ($synonyms as $key => $value) {
    echo "$key is a synonym for $value.\n";
}
```

As you can see, the `foreach` loop reuses the same syntax that's used to create associative arrays.

`for` Loops

Use `for` loops when you want to run a loop a specific number of times. The loop statement has three parts: a variable assignment for the loop's counter, an expression (containing the index variable) that specifies when the loop should stop running, and an expression that increments the loop counter. Here's a typical `for` loop:

```
for ($i = 1; $i <= 10; $i++)
{
  echo "Loop executed $i times.\n";
}
```

`$i` is the counter (or index variable) for the loop. The loop is executed until `$i` is larger than 10 (meaning that it will run 10 times). The last expression, `$i++`, adds one to `$i` every time the loop executes. The `for` loop can also be used to process an array instead of `foreach` if you prefer. You just have to reference the array in the loop statement, like this:

```
$colors = array('red', 'green', 'blue');
for ($i = 0; $i < count(array); $i++) {
  echo "Currently processing " . $colors[$i] . ".\n";
}
```

There are a couple of differences between this loop and the previous one. In this case, I start the index variable at 0, and use < rather than <= as the termination condition for the loop. That's because `count()` returns the size of the `$colors` array, which is 3, and loop indexes start with 0 rather than 1. If I start at 0 and terminate the loop when `$i` is equal to the size of the `$colors` array, it runs three times, with `$i` being assigned the values 0, 1, and 2, corresponding to the indexes of the array being processed.

while and do...while Loops

Both for and foreach are generally used when you want a loop to iterate a specific number of times. The while and do...while loops, on the other hand, are designed to be run an arbitrary number of times. Both loop statements use a single condition to determine whether the loop should continue running. Here's an example with while:

```
$number = 1;
while ($number != 5) {
  $number = rand(1, 10);
  echo "Your number is $number.\n";
}
```

This loop runs until $number is equal to 5. Every time the loop runs, $number is assigned a random value between 1 and 10. When the random number generator returns a 5, the while loop will stop running. A do...while loop is basically the same, except the condition appears at the bottom of the loop. Here's what it looks like:

```
$number = 1;
do {
  echo "Your number is $number.\n";
  $number = rand(1, 10);
} while ($number != 5);
```

Generally speaking, the only time it makes sense to use do … while is when you want to be sure the body of the loop will execute at least once.

Controlling Loop Execution

Sometimes you want to alter the execution of a loop. Sometimes you need to stop running the loop immediately, and other times you might want to just skip ahead to the next iteration of the loop. Fortunately, PHP offers statements that do both. The break statement is used to immediately stop executing a loop and move on to the code that follows it. The continue statement stops the current iteration of the loop and goes straight to the loop condition.

Here's an example of how break is used:

```
$colors = ('red', 'green', 'blue');
$looking_for = 'red';
foreach ($colors as $color) {
  if ($color = $looking_for) {
    echo "Found $color.\n";
    break;
  }
}
```

In this example, I'm searching for a particular color. When the `foreach` loop gets to the array element that matches the color I'm looking for, I print the color out and use the `break` statement to stop the loop. When I've found the element I'm looking for, there's no reason to continue.

I could accomplish the same thing a different way using `continue`, like this:

```
$colors = ('red', 'green', 'blue');
$looking_for = 'red';
foreach ($colors as $color) {
  if ($color != $looking_for) {
    continue;
  }

  echo "Found $color.\n";
}
```

In this case, if the color is not the one I'm looking for, the `continue` statement stops executing the body of the loop and goes back to the loop condition. If the color is the one I'm looking for, the `continue` statement is not executed and the `echo` function goes ahead and prints the color name I'm looking for.

The loops I'm using as examples don't have a whole lot of work to do. Adding in the `break` and `continue` statements doesn't make my programs much more efficient. Suppose, however, that each iteration of my loop searches a very large file or fetches some data from a remote server. If I can save some of that work using `break` and `continue`, it could make my script much faster.

Built-in Functions

PHP supports literally hundreds of built-in functions. You've already seen a few, such as `echo()` and `count()`. There are many, many more. PHP has functions for formatting strings, searching strings, connecting to many types of databases, reading and writing files, dealing with dates and times, and just about everything in between.

You learned that most of the functionality in the JavaScript language is built using the methods of a few standard objects such as `window` and `document`. PHP is different—rather than its built-in functions being organized via association with objects, they are all just part of the language's vocabulary.

If you ever get the feeling that there might be a built-in function to take care of some task, check the PHP manual to see whether such a function already exists. Chances are it does. Definitely check whether your function will manipulate strings or arrays. PHP has a huge library of array- and string-manipulation functions that take care of most common tasks.

User-Defined Functions

PHP enables you to create user-defined functions that, like JavaScript functions, enable you to package up code you want to reuse. Here's how a function is declared:

```
function myFunction($arg = 0) {
  // Do stuff
}
```

The `function` keyword indicates that you're creating a user-defined function. The name of the function follows. In this case, it's `myFunction`. The rules for function names and variable names are the same—numbers, letters, and underscores are valid. The list of arguments that the function accepts follows the function name, in parentheses.

The preceding function has one argument, `$arg`. In this example, I've set a default value for the argument. The variable `$arg` would be set to 0 if the function were called like this:

```
myFunction();
```

On the other hand, `$arg` would be set to 55 if the function were called like this:

```
myFunction(55);
```

Functions can just as easily accept multiple arguments:

```
function myOtherFunction($arg1, $arg2, $arg3)
{
  // Do stuff
}
```

As you can see, `myOtherFunction` accepts three arguments, one of which is an array. Valid calls to this function include the following:

```
myOtherFunction('one', 'two', array('three'));
myOtherFunction('one', 'two');
myOtherFunction(0, 0, @stuff);
myOtherFunction(1, 'blue');
```

One thing you can't do is leave out arguments in the middle of a list. So if you have a function that accepts three arguments, there's no way to set just the first and third arguments and leave out the second, or set the second and third and leave out the first. If you pass one argument in, it will be assigned to the function's first argument. If you pass in two arguments, they will be assigned to the first and second arguments to the function.

Returning Values

Optionally, your function can return a value, or more specifically, a *variable*. Here's a simple example of a function:

```
function add($a = 0, $b = 0) {
  return $a + $b;
}
```

The `return` keyword is used to indicate that the value of a variable should be returned to the caller of a function. You could call the previous function like this:

```
$sum = add(2, 3); // $sum set to 5
```

A function can just as easily return an array. Here's an example:

```
function makeArray($a, $b) {
  return array($a, $b);
}

$new_array = makeArray('one', 'two');
```

If you don't explicitly return a value from your function, PHP will return the result of the last expression inside the function anyway. For example, suppose I wrote the `add` function like this:

```
function add($a = 0, $b = 0) {
 $a + $b;
}
```

Because `$a + $b` is the last expression in the function, PHP will go ahead and return its result. That's the case for logical expressions, too. Here's an example:

```
function negate($a) {
  !$a;
}

negate(1); // returns false
negate(0); // returns true
```

Your function can also return the result of another function, whether it's built in or one you wrote yourself. Here are a couple of examples:

```
function add($a = 0, $b = 0) {
  return $a + $b;
}
```

```
function alsoAdd($a = 0, $b = 0) {
  return add($a, $b);
}
```

Processing Forms

You learned how to create forms back in Lesson 11, "Designing Forms," and although I explained how to design a form, I didn't give you a whole lot of information about what to do with form data once it's submitted. Now I explain how PHP makes data that has been submitted available to your PHP scripts.

When a user submits a form, PHP automatically decodes the variables and copies the values into some built-in variables. Built-in variables are like built-in functions—you can always count on their being defined when you run a script. The three associated with form data are `$_GET`, `$_POST`, and `$_REQUEST`. These variables are all associative arrays, and the names assigned to the form fields on your form are the keys to the arrays.

`$_GET` contains all the parameters submitted using the `GET` method (in other words, in the query string portion of the URL). The `$_POST` method contains all the parameters submitted via `POST` in the response body. `$_REQUEST` contains all the form parameters regardless of how they were submitted. Unless you have a specific reason to differentiate between `GET` and `POST`, you can use `$_REQUEST`. Let's look at a simple example of a form:

```
<form action="post.php" method="post">
  Enter your name: <input type="text" name="yourname" /><br />
  <input type="submit" />
</form>
```

When the user submits the form, the value of the `yourname` field will be available in `$_POST` and `$_REQUEST`. You could return it to the user like this:

```
<p>Hello <?= $_REQUEST['yourname'] ?>. Thanks for visiting.</p>
```

Preventing Cross-Site Scripting

You have to be careful when you display data entered by a user on a web page because malicious users can include HTML tags and JavaScript in their input in an attempt to trick other users who might view that information into doing something they might not want to do, such as entering their password to your site and submitting it to another site. This is known as a *cross-site scripting attack*.

To prevent malicious users from doing that sort of thing, PHP includes the `htmlspecialchars()` function, which will automatically encodes any special characters in a string so that they are displayed on a page rather than letting the browser treat

them as markup. Or if you prefer, you can use `htmlentities()`, which encodes all the characters that are encoded by `htmlspecialchars()` plus any other characters that can be represented as entities. In the preceding example, you really want to write the script that displays the user's name like this:

```
<p>Hello <?= htmlspecialchars($_POST['yourname']) ?>.
Thanks for visiting.</p>
```

That prevents the person who submitted the data from launching a successful cross-site scripting attack.

If you prefer, you can also use the `strip_tags()` function, which just removes all the HTML tags from a string.

Finally, if your form is submitted using the `POST` method, you should refer to the parameters using `$_POST` rather than `$_REQUEST`, which also helps to avoid certain types of attacks by ignoring information appended to the URL via the query string.

When you have access to the data the user submitted, you can do whatever you like with it. You can validate it (even if you have JavaScript validation, you should still validate user input on the server as well), store it in a database for later use, or send it to someone via email.

Handling Parameters with Multiple Values

Most form fields are easy to deal with; they're simple name and value pairs. If you have a text field or radio button group, for example, you can access the value submitted using `$_REQUEST`, like this:

```
$radio_value = $_REQUEST['radiofield'];
$text_value = $_REQUEST['textfield'];
```

However, some types of fields submit multiple name and value pairs, specifically check boxes and multiple select lists. If you have a group of five check boxes on a form, that field can actually submit up to five separate parameters, all of which have the same name and different values. PHP handles this by converting the user input into an array rather than a regular variable. Unfortunately, you have to give PHP a hint to let it know that a field should be handled this way. (PHP has no idea what your form looks like; all it knows about is the data that has been submitted.)

If you include `[]` at the end of the name of a form field, PHP knows that it should expect multiple values for that field and converts the parameters into an array. This occurs even if only one value is submitted for that field. Here's an example:

```
<form action="postmultiplevalues.php" method="post">
  <input type="checkbox" name="colors[]" value="red" /> Red<br />
  <input type="checkbox" name="colors[]" value="green" /> Green<br />
  <input type="checkbox" name="colors[]" value="blue" /> Blue
</form>
```

When the form is submitted, you can access the values as you would for any other parameter, except that the value in the `$_REQUEST` array for this parameter will be an array rather than a single value. You can access it like this:

```
$colors = $_REQUEST['colors'];
foreach ($colors as $color) {
  echo "$color<br />\n";
}
```

If the user selects only one check box, the value will be placed in an array that has only one element.

▼ Task: Exercise 21.1: Validating a Form

One of the most common tasks when it comes to server-side processing is form validation. When users submit data via a form, it should be validated on the server, even if your page includes JavaScript validation, because you can't guarantee that JavaScript validation was actually applied to the form data.

I use a simplified version of the user registration form from Lesson 11 in this exercise. Figure 21.1 is a screenshot of the form I'll be using. Here's the HTML source:

Input ▼

```
<!DOCTYPE html>
<html>
<head>
<title>Registration Form</title>
</head>
<body>
<h1>Registration Form</h1>

<p>Please fill out the form below to register for our site. Fields
with bold labels are required.</p>

<form method="post">

<p><label for="name"><b>Name:</b></label><br />
<input name="name" /></p>
```

```
<p><label for="age"><b>Age:</label></b><br />
<input name="age" /></p>

<p><b>Toys:</b><br />
<label><input type="checkbox" name="toys[]" value="digicam" /> Digital
Camera</label><br />
<label><input type="checkbox" name="toys[]" value="mp3" /> MP3 Player</label><br
/>
<label><input type="checkbox" name="toys[]" value="wlan" /> Wireless
LAN</label></p>

<p><input type="submit" value="register" /></p>
</form>
</body>
</html>
```

Output ▶

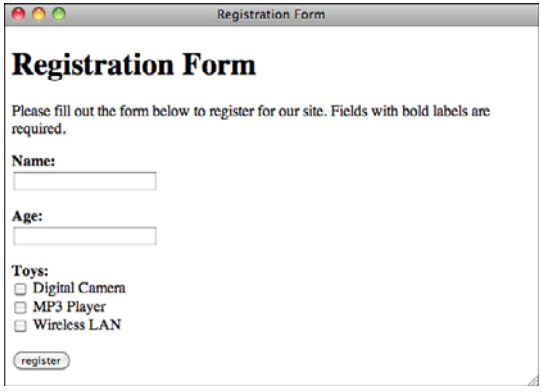


FIGURE 21.1
A simple user registration form.

As you can see, the form has three fields: one for the user's name, one for the user's age, and one that enables the user to select some toys he or she owns. All three of the fields are required. The form submits to itself, using the `POST` method. I've specified the action for the form using a built-in PHP variable that returns the URL for the page currently being displayed. That way I can make sure the form is submitted to itself without including the URL for the page in my HTML. Here's the basic structure of the page:

```
<?php
// Form processing code
?>
<html>
    <head>
        <title>Page Structure</title>
        <style type="text/css">
            /* Page styles go here. */
        </style>
```

```
    </head>
    <body>
        <h1>Sample Page</h1>
        <!— Print form errors here —>
        <form method="post" action="<?= $_SERVER['PHP_SELF'] ?>">
            <!— Present form fields here —>
        </form>
    </body>
</html>
```

This structure is pretty common for pages that present a form and process that form, too. The PHP processor runs the scripts on the page from top to bottom, so all the form processing will take place before any of the page is presented. If this page were going to do more than just validate the form, it would probably redirect the user to a page thanking her for registering if the validation code found no errors. It would also probably save the values submitted through the form somewhere. In this case, though, I'm just explaining form validation.

As you can see, the form-processing code lives on the same page as the form itself, so the form will be submitted to this page. The validation code will live within the script section at the top of the page. My objective for this page is to make sure that the user enters all the required data and that the age the user enters is actually a number. To make things a bit easier on myself, I've written a function to do the actual validation for me.

Here's the function:

```
function validate() {
    $errors = array();

    if (empty($_POST['name'])) {
        $errors['name'] = 'You must enter your name.';
    }

    if (!is_numeric($_POST['age'])) {
        $errors['age'] = "You must enter a valid age.";
    }

    if (empty($_POST['toys'])) {
        $errors['toys'] = 'You must choose at least one toy.';
    }

    return $errors;
}
```

This function validates each of the fields on the form and then places all the errors in an associative array called `$errors`. When an error is detected, a new entry is added to the

array with the name of the field as the key and the error message as the array value. Later on, I display the error messages and use the field names to mark the fields that have errors.

On the first line of the function, I declare `$errors` to store the errors found during validation. Next, I validate the `name` parameter. PHP has a built-in function called `empty()` that checks to see whether a variable is empty. In this case, I use it to check `$_POST['name']`, which was set automatically when the form was submitted. If that variable is empty, meaning that the user did not submit his or her name, I add an entry to `$errors`.

Next, I validate the `age` field using PHP's `is_numeric()` function. I negate the condition with the not operator because it's only an error if the value in the field isn't a number.

Finally, I check to make sure that the user has selected a toy. As you saw, this field is actually a check box group, meaning that the contents of the field are submitted as an array (assuming I've named the field properly). Again, I use `empty()` here. It works with regular variables and arrays, and it returns true if an array contains no elements. If there are no elements in the array, no toys were submitted, and the error is added to the array.

When validation is complete, I return the value of the `$errors` variable to the caller. Here's the code I use to call the `validate()` function. It lives right at the top of the page:

```
$errors = array();

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $errors = validate();
}
```

I check on `$errors` later in the page regardless of whether I validate the input, so I go ahead and declare it. To determine whether I should validate a form submission or display an empty form, I check the built-in variable `$_SERVER['REQUEST_METHOD']` to see whether the request was submitted using the `POST` method. If it was, I want to do the input validation. If not, I just want to display the form.

If any parameters were submitted via `POST`, I run the `validate()` function I just described. There's one more line of code at the top of the page where most of my PHP code lives:

```
$toys = array('digicam' => 'Digital Camera',
    'mp3' => 'MP3 Player', 'wlan' => 'Wireless LAN');
```


It's an array that contains a list of all the check boxes to display for the `toys` field. It's easier to iterate over all the check boxes in a loop than it is to code them all by hand. If I

need to add new toys to the list, I can just add them to the array definition and they'll automatically be included on the page. I show you how the code that displays the field works shortly.

Presenting the Form

Aside from validating form submissions, one of the other important functions of server-side processing is to prepopulate forms with data when they are presented. Many web applications are referred to as *CRUD applications*, where CRUD stands for create/update/delete. It describes the fact that the applications are used to mostly manage records in some kind of database. If a user submits a form with invalid data, when you present the form for the user to correct, you want to include all the data that the user entered so that he doesn't have to type it all in again. By the same token, if you're writing an application that enables users to update their user profile for a website, you will want to include the information in their current profile in the update form. This section explains how to accomplish these sorts of tasks.

However, before I present the form to the user, I provide a list of errors that the user needs to correct before the form submission is considered valid. Here's the code to accomplish that task:

```
<?php if (!empty($errors)) { ?>
    <ul>
        <?php foreach (array_values($errors) as $error) { ?>
            <li><?= $error ?></li>
        <?php } ?>
    </ul>
<?php } ?>
```

I use the `empty()` function yet again to determine whether there are any errors. If there aren't any, I can go ahead and present the form. If there are any errors, I present them in a list. First, I create an unordered list; then I use a `foreach` loop to iterate over the errors. The `$errors` variable is an associative array, and the error messages to present are the values in the array. I use the built-in `array_values()` function to extract an array containing only the values in the `$errors` array, and iterate over that array using the `foreach` loop. There's something interesting going on here. The body of the `foreach` loop is HTML, not PHP code. Look closely and you'll see the opening and closing braces for the `foreach` loop. Instead of sticking with PHP for the body of the loop, though, I go back to HTML mode by closing the PHP script, and I use regular HTML to define the list items. Inside the list item I use a short tag to present the current error message.

Separating Presentation and Logic

The point here is that it's common to mix PHP and HTML in this way. You create your loop using PHP but you define the HTML in the page rather than in `echo()` calls inside your PHP code. This is generally considered the best practice for PHP. You should write as much HTML as possible outside your PHP scripts, using PHP only where it's necessary to add bits of logic to the page. Then you can keep the bulk of your PHP code at the top or bottom of your page or in included files to separate the presentation of your data and the business logic implemented in code. That makes your code easier to work on in the future. As an example, rather than sprinkling the validation code throughout my page, I put it in one function so that a programmer can work on it without worrying about the page layout. By the same token, I could have built the unordered list inside the validation function and just returned that, but then my HTML would be mixed in with my PHP. Cleanly separating them is generally the best approach.

After I've listed the errors, I can go ahead and present the form fields. Before I do that, let me show you one more thing I've added to the page. I included a style sheet that defines one rule: `label.error`. The labels for any fields with errors will be assigned to this class so that they can be highlighted when the form is presented. Here's the style sheet:

```
<style type="text/css">
label.error {
    color: red;
}
</style>
```

Okay, now that everything is set up, let's look at how the `name` field is presented. Here's the code:

```
<p>
<?php if (array_key_exists('name', $errors)) { ?>
    <label for="name" class="error"><b>Name:</b></label>
<?php } else { ?>
    <label for="name"><b>Name:</b></label>
<?php } ?>
<br />
<input name="name" value="<?= strip_tags($_POST['name']) ?>" /></p>
```

This code is a lot different from the old code I used to present the `name` field in the original listing in this example. First, I include an `if` statement that checks to see whether

there's an error associated with this field. To do so, I use the `array_key_exists()` function, which is yet another built-in PHP function. Remember that the keys in `$errors` are the names of the fields with errors. So if the `$errors` array contains an element with the key `name`, it means that this field was not valid.

If there is an error with the field, I include the attribute `class="error"` in the `<label>` tag for the field. When the form is presented, the label will be red, indicating to the user that she needs to fix that field, even if the user didn't bother to read the list of errors. If there isn't an error, the normal `<label>` tag is printed.

When that's done, I just have to print out the `name` field, but in this case I need to include the value that was submitted for the field if it's available. I include the `value` attribute in my `<input>` tag, and I use a short tag to include the value of `$_POST['name']` as the value. Inside the expression evaluator, I've wrapped the variable containing the value for the field in the `strip_tags()`function. This PHP function automatically removes any HTML tags from a string, and it's used here to thwart any possible cross-site scripting attacks a malicious person might employ.

The `age` field is identical to the `name` field in every way except its name, so I skip that and turn instead to the `toys` field. Here's the code:

```
<p>
<?php if (array_key_exists('toys', $errors)) { ?>
    <label for="toys[]" class="error"><b>Toys:</b></label>
<?php } else { ?>
    <label for="toys[]"><b>Toys:</b></label>
<?php } ?>
<br />
<?php foreach ($toys as $key => $value) { ?>
    <label><input type="checkbox" name="toys[]"
    <?php if (in_array($key, $_POST['toys'])) { echo 'checked="checked" '; } ?>
    value="<?= $key ?>" /> <?= $value?></label><br />
<?php } ?>
</p>
```

As you can see, the code for marking the label for the field as an error is the same for this field as it was for `name`. The more interesting section of the code here is the loop that creates the check boxes. When I was describing the form-processing section of the page, I explained that I put all the toys inside the array `$toys` so that I could print out the check boxes using a loop. There's the loop.

The values in the `<input>` tags are the keys in the array, and the labels for the check boxes are the values. I use the associative array version of the `foreach` loop to copy each of the key/value pairs in the array into the variables `$key` and `$value`. Inside the loop, I

print out the <input> tags, using toys[] as the parameter name to let PHP know that this field can have multiple values and should be treated as an array. To include the value for a field, I just use a short tag to insert the key into the value attribute of the tag. I use it again to print out the label for the check box, $value, after the tag. The last bit here is the if statement found within the <input> tag. Remember that if you want a check box to be prechecked when a form is presented, you have to include the checked attribute. I use the in_array() function to check if the key currently being processed is in $_POST['toys']. If it is, I then print out the checked attribute using echo(). This ensures that all the items the user checked before submitting the form are still checked if validation fails.

A browser displaying a form that contains some errors appears in Figure 21.2. Here's the full source listing for the page:

Input

```
<?php
    $toys = array('digicam' => 'Digital Camera',
        'mp3' => 'MP3 Player', 'wlan' => 'Wireless LAN');

    $errors = array();

    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $errors = validate();
    }

    function validate() {
        $errors = array();

        if (empty($_POST['name'])) {
            $errors['name'] = 'You must enter your name.';
        }

        if (!is_numeric($_POST['age'])) {
            $errors['age'] = "You must enter a valid age.";
        }

        if (empty($_POST['toys'])) {
            $errors['toys'] = 'You must choose at least one toy.';
        }

        return $errors;
    }
?>
<!DOCTYPE html>
<html>
<head>
```

```
<title>Registration Form</title>
<style type="text/css">
label.error {
    color: red;
}
</style>
</head>
<body>
<h1>Registration Form</h1>

<p>Please fill out the form below to register for our site. Fields
with bold labels are required.</p>

<?php if (!empty($errors)) { ?>
    <ul>
        <?php foreach (array_values($errors) as $error) { ?>
            <li><?= $error ?></li>
        <?php } ?>
    </ul>
<?php } ?>

<form method="post" action="<?= $_SERVER['PHP_SELF'] ?>">

<p>
<?php if (array_key_exists('name', $errors)) { ?>
    <label for="name" class="error"><b>Name:</b></label>
<?php } else { ?>
    <label for="name"><b>Name:</b></label>
<?php } ?>
<br />
<input name="name" value="<?= strip_tags($_POST['name']) ?>" /></p>

<p>
<?php if (array_key_exists('age', $errors)) { ?>
    <label for="age" class="error"><b>Age:</b></label>
<?php } else { ?>
    <label for="age"><b>Age:</b></label>
<?php } ?>
<br />
<input name="age" value="<?= strip_tags($_POST['age']) ?>"/></p>

<p>
<?php if (array_key_exists('toys', $errors)) { ?>
    <label class="error"><b>Toys:</b></label>
<?php } else { ?>
    <label><b>Toys:</b></label>
<?php } ?>
<br />
<?php foreach ($toys as $key => $value) { ?>
    <label><input type="checkbox" name="toys[]"
    <?php if (array_key_exists('toys', $_POST) && in_array($key,
```

```
$_POST['toys'])) { echo 'checked="checked" '; } ?>
    value="<?= $key ?>" /> <?= $value ?></label><br />
<?php } ?>
</p>

<p><input type="submit" value="register" /></p>
</form>
</body>
</html>
```

Output ▶

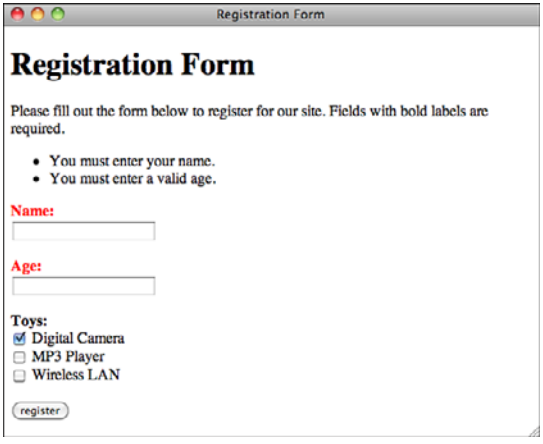


FIGURE 21.2
A form with some errors that were caught during validation.

Using PHP Includes

PHP and all other server-side scripting languages provide the ability to include snippets of code or markup in pages. With PHP, the ability to include files is built into the language. Because the include statements are part of the language, you don't need to include parentheses around the name of the file to be included. You can conditionally include files, specify which file to include dynamically, or even nest include function calls within included pages. Here's a simple example of an include call:

```
include "header.php";
```

On encountering that function call, PHP will try to read in and process a file named `header.php` in the same directory as the current page. If it can't find this file, it will try to find the file in each of the directories in its *include path*, too. The include path is a list of directories (generally specified by the server administrator) where PHP searches for files to include, and it's generally set for the entire server in a configuration file.

Four include-related functions are built in to PHP: `require`, `require_once`, `include`, and `include_once`. All these functions include an external file in the page being processed. The difference between `include` and `require` is how PHP reacts when the file being included isn't available. If `include` or `include_once` is used, the PHP page prints a warning and continues on. If `require` or `require_once` is used, an unavailable include file is treated as a fatal error and page processing stops.

If you use `require_once` or `include_once` to include a file that was already included on the page, the function call will be ignored. If you use `require` or `include`, the file will be included no matter what.

PHP includes are like HTML links in that you can use relative or absolute paths in your includes. The difference is that absolute PHP paths start at the root of file system rather than the web server's document root. So if you want to include a file using an absolute path on a computer running Windows, you write the include like this:

```
require_once 'c:\stuff\myfile.php';
```

That's almost never a good idea. You should always use relative paths where possible. In other words, if the included file is in the directory above the one where the including file is located, you should use a path like this:

```
require_once "../myinclude.php";
```

If the file being included is not stored with your other web documents, try to have that directory added to your server's include path rather than using absolute paths to access it.

CAUTION

Never pass data entered by a user to any include function; it's a big security risk. For example, this would be inappropriate:

```
require_once $_POST['file_to_include';
```

PHP includes can be useful even if you don't plan on doing any programming in PHP. You can turn parts of your website that you use frequently into files to be included, saving you from having to edit the same content in multiple places when you're working on your site. Using PHP includes this way can provide the same advantages that putting your CSS and JavaScript into external files does. For example, you might create a file called `header.php` that looks like this:

```
<!DOCTYPE html>
<html>
<head>
    <title><?= $title ?></title>
    <script src="site.js"></script>
```

```
    <link rel="stylesheet" href="site.css">
</head>
<body>
```

This file includes all the tags for the start of my page, including links to external JavaScript and CSS files. There's a PHP short tag in the title that prints out the value of the `$title` variable. That enables you to use the header file for all of your pages and to specify individual titles for each of them. To include this file, you use the following code:

```
<?php
$title = "Welcome!";
include "header.php";
?>
```

Choosing Which `Include` Function to Use

Given these four very similar functions, how do you choose which makes the most sense to use? The most important factor in making that decision is the content of the file to be included. Generally, there are two types of include files: snippets of markup that will be presented on your page, and PHP code libraries that provide code you are using on multiple pages throughout a site.

If the file you are including is a library, you just about always want to use `require_once`. If you're using code from the library on a page, chances are the page will not work if the library file is not available, meaning that you should use `require` rather than `include`. If the file contains library code, you're not going to want to include it more than once. Let's look at an example. You've written a library called `temperature_converter.php`. The contents of the file are shown here:

```
<?php
function celsiusToFahrenheit($temp = 0) {
    return round(($temp * 9/5) + 32);
}
?>
```

This file contains one function, `celsiusToFahrenheit()`, which converts a Celsius temperature to Fahrenheit and then rounds the result so that the function returns an integer. Now let's look at a page that includes this file:

```
<?php
require_once "temperature_converter.php";
?>
<html>
  <head>
```

```
    <title>Current Temperature</title>
  </head>
  <body>
  <p>Current temperature in Fahrenheit: <?= celsiusToFahrenheit(55) ?></p>
  </body>
</html>
```

As you can see, in this case the page won't have any meaning if the function in the library page is not available, so using `require` makes sense. On this page, it wouldn't matter whether I used `require` or `require_once` because there are no other includes. Suppose that the page included another file, one that prints the current temperatures around the world. If that page also had a `require()` call for `temperature_converter.php`, the same code would be included twice. An error would cause the page to fail, because each function name can only be declared once. Using `require_once` ensures that your library code is available and that it is not accidentally included in your page multiple times.

On the other hand, if you're including content that will be displayed within your page, then `include` or `require` make more sense. You don't have to worry about conflicts, and if you're including something to be displayed on the page, chances are you want it to appear, even if you've already included the same thing.

Expanding Your Knowledge of PHP

PHP is a full-featured scripting language for creating web applications and even writing command-line scripts. What you've seen in this lesson is just a brief introduction to the language. There are more statements, lots more built-in functions, and plenty of other things about the application for which there isn't space to discuss in this lesson. Fortunately, an online version of the PHP manual is available that will fill in most of the blanks for you. You can find it at http://www.php.net/docs.php.

Also, shelves of books about PHP are available to you. Some that you might want to look into are *Sams Teach Yourself PHP, MySQL, and Apache All in One* (ISBN 067232976X), and *PHP and MySQL Web Development* (ISBN 0672317842).

There's more to PHP than just the core language, too. Lots of libraries have been written by users to take care of common programming tasks that you might run into. There's an online repository for these libraries called PEAR, which stands for PHP Extension and Application Repository. You can find it at http://pear.php.net/.

For example, the eBay website provides an API (application programming interface) that you can use to integrate your own website with eBay. You could write the code to use this API yourself, but a library in PEAR already exists. You can find it at http://pear.php.net/package/Services_Ebay.

This is just one of the many libraries you can obtain via PEAR. When you're writing your applications, make sure to check the PHP manual to ensure there's not already a built-in function to take care of whatever you're doing. If there isn't, check PEAR.

As I said before, I left out huge swaths of PHP functionality in this lesson for the sake of space. Here are some areas that you'll want to look into before developing your own PHP applications.

Database Connectivity

I mentioned CRUD applications already. A CRUD application is generally just a front end for a relational database, which in turn is an application optimized for storing data within tables. Databases can be used to store content for websites, billing information for an online store, payroll for a company, or anything else that can be expressed as a table. It seems like there's a relational database providing the storage for just about every popular website.

Because databases play such a huge role in developing web applications, PHP provides a lot of database-related functionality. Most relational databases are applications that can be accessed over a network, a lot like a web server. PHP is capable of connecting to every popular relational database. To communicate with relational databases, you have to use a language called SQL (the Structured Query Language). That's another book unto itself.

Regular Expressions

Regular expressions comprise a small language designed to provide programmers with a flexible way to match patterns in strings. For example, the regular expression `^a.*z$` matches a string that starts with *a*, ends with *z*, and has some number of characters in between. You can use regular expressions to do much more fine-grained form validation than I did in Exercise 21.1. They're also used to extract information from files, search and replace within strings, parse email addresses, or anything else that requires you to solve a problem with pattern matching. Regular expressions are incredibly flexible, but the syntax can be a bit complex.

PHP actually supports two different varieties of regular expression syntax: Perl style and POSIX style. You can read about both of them in the PHP manual.

Sending Mail

PHP provides functions for sending email. For example, you could write a PHP script that automatically notifies an administrator by email when a user registers for a website, or sends users a password reminder if they request one when they forget their password. PHP also provides functions that enable your applications to retrieve mail as well as send it, making it possible to write web-based email clients and other such applications.

Object-Oriented PHP

PHP provides features for object-oriented development if you prefer that style of programming. For more information on object-oriented PHP, refer to the manual.

Cookies and Sessions

Cookies are a browser feature that lets websites set values that are stored by your browser and returned to the server any time you request a page. For example, when users log in to your site, you can set a cookie on their computers to keep track of who they are so that you don't have to force them to log in any time they want to see a password-protected page. You can also use cookies to keep track of when visitors return to your site after their initial visit. PHP provides full support for cookies. It also provides a facility called *sessions*. Sessions enable you to store data between requests to the server. For example, you could read a user's profile into her session when that user logs into the site, and then reference it on every page without going back and loading it all over again. Generally, cookies are used with sessions so that the server can keep track of which session is associated with a particular user.

File Uploads

In Lesson 11, "Designing Forms," you learned about file upload fields for forms. PHP can deal with file uploads, enabling the programmer to access and manipulate them. With PHP, file uploads are stored to a temporary location on the server, and it's up to the programmer to decide whether to store them permanently and, if so, where to put them.

Other Application Platforms

PHP is just one of many programming languages that people use to write web applications. It is the language used to create popular web applications like Drupal, WordPress, and Expression Engine. It's also the tool used by major web companies like Facebook and Yahoo! However, other options are available. If you're just diving into web programming, PHP is probably a good choice, but you might find yourself working on applications written in another language. Here's a brief overview of the languages you may encounter.

Microsoft ASP.NET

Microsoft provides the ASP.NET environment for writing web applications that run on Windows servers. ASP.NET is similar to PHP in that it supports embedding server-side code in HTML pages. It supports Visual Basic and C# as programming languages and runs on Microsoft's Internet Information Server, which is included with Windows Server. You can read more about ASP.NET and download free tools for developing and running ASP.NET applications at http://www.asp.net/.

Java EE

Java is a programming language originally created by Sun that runs on many operating systems, including Windows, OS X, and Linux. EE stands for Enterprise Edition, an umbrella under which the server-side Java technologies live. Java is widely used by large companies to build internal and external applications.

There are two ways to write web applications in Java—servlets, which are programs that run on the server and can produce web content as output; and Java Server Pages, which allow you to embed Java code in HTML pages so that it can be executed on the server. You can read more about it at http://java.sun.com/javaee/.

Ruby on Rails

Ruby on Rails is a newer application platform that is gaining popularity because it enables developers to get a lot done with just a few lines of code. It uses the Ruby programming language and is designed with the philosophy that applications can be written quite efficiently if developers adhere to the conventions that the creators of the Ruby on Rails framework built in to it. You can read more about it at http://rubyonrails.org.

Summary

This lesson provided a whirlwind tour of the PHP language, and it explained how server-side scripts are written in general. Although the syntax of other languages will differ from PHP, the basic principles for dealing with user input, processing forms, and embedding scripts in your pages will be quite similar. I also listed some other application platforms you might encounter. They are all similar to PHP in function, even though the syntax of the languages they use differ from PHP to varying degrees.

In the next lesson, you learn how to take advantage of applications that other people have written rather than writing them yourself. Just as PHP has lots of built-in functions to take care of common tasks, so too are there many popular applications that you can download and install rather than writing them from scratch yourself.

Workshop

The following workshop includes questions you might ask about server-side development, quizzes to test your knowledge, and three quick exercises.

Q&A

Q At work, all of our applications are written using Active Server Pages. Why didn't you write about that?

A There are a number of popular platforms for writing web applications. PHP has the advantage of running on a number of operating systems, including Windows, Mac OS X, and Linux. Furthermore, support for PHP is offered by many web hosting providers. Finally, as you'll learn in the next lesson, there are many applications already written in PHP that you can take advantage of. Knowledge of PHP can be helpful in working with them.

Q Do I need a special application to edit PHP files?

A Just as with HTML, PHP files are normal text documents. Some text editors have specialized features that make working with PHP easier, just as there are for HTML. If you're just starting out, using Notepad or any other regular text editor will work fine, but you'll probably want to find a more powerful tool for writing PHP if you find yourself programming in PHP a lot.

Q How do I deploy PHP files to a server?

A There are no special requirements for deploying PHP files. You can just transfer them to the server as you would regular HTML files. As long as the server is configured to handle PHP, you should be fine. The one thing you do need to be careful to do is to make sure your directory structure is the same on the server and on your local computer. If you are using includes and directory paths change, your includes will break.

Q Are PHP scripts browser dependent in any way?

A All the processing in PHP scripts takes place on the server. They can be used to produce HTML or JavaScript that won't work with your browser, but there's nothing in PHP that will prevent it from working with a browser.

Quiz

1. What is the difference between double and single quotes in PHP?
2. How do the `include_once` and `require_once` functions differ?
3. Which functions can be used to help avoid cross-site scripting attacks?
4. How do you declare an associative array in PHP?

Quiz Answers

1. In PHP, strings in double quotes are parsed for variable references and special characters before they are presented. Strings in single quotes are presented as is.
2. The `include_once` function does not return a fatal error if the file being included is not found. With `require_once`, if the file is not found, a fatal error occurs and the rest of the page is not processed.
3. You can use `htmlspecialchars()` to escape the characters used to generate HTML tags for a page. You can use `strip_tags()` to remove any HTML tags from a string. Either approach should prevent users from using malicious input to attempt a cross-site scripting attack.
4. Associative arrays are declared as follows:

```
$array = ('key' => 'value, 'key2' => 'value2');
```

Exercises

1. Get PHP up and running on your own computer.
2. Write a script that enables a user to show the current date and time on a web page.
3. Go to the PHP manual online and find a built-in function that wasn't introduced in this lesson. Use it in a script of your own.

This page intentionally left blank

LESSON 22

Content Management Systems and Publishing Platforms

Throughout the course of this book, you've learned how to build websites by hand, creating HTML pages one by one. In this lesson, you learn how to use applications to manage aspects of your website. These types of applications are generally called *content management systems*, and I explain how to use them to keep track of your content so that it's easier to publish and maintain.

In this lesson, you learn the following:

- How content management systems came into being
- How to decide whether to use a content management system for your website
- What the common types of content management systems are
- How to work with hosted web applications in general
- How to use four popular content management applications: WordPress, Drupal, TypePad, and MediaWiki

The Rise of Content Management

Content management systems were invented to deal with the complexity and tedium of managing large collections of HTML documents by hand. A content management system is an application that manages website content in a form that's convenient for editing and maintenance and publishes the content as HTML. Some content management systems generate HTML programmatically on-the-fly, whereas others generate static files and publish them. Most content management systems are web applications themselves. Users enter and manage content via HTML forms, and the content management system takes care of storing the content in a database of some kind. Generally, content management systems provide some kind of facility for creating templates and then merge the content entered by users with the templates to produce a finished website. When templates are updated, all the pages that use those templates automatically have the changes applied to them.

Content management systems were initially deployed at the largest sites, such as online news sites, online catalogs, and technical sites that published lots of data. These systems were generally custom built and rather complicated. Content management eventually became an industry unto itself, with many companies producing large, expensive systems aimed at publishers, large corporate websites, and other customers who had lots of web content and a few hundred thousand dollars to spend to deploy a system to manage that content.

At the same time, content management systems aimed at individuals also became increasingly popular, except that they weren't called content management tools. Instead, they were called weblogging (blogging) tools, wikis, photo galleries, and so on.

These days, most websites are built using content management systems of some kind, and some of the largest and busiest websites utilize content management tools that were originally aimed at individuals running small sites. For example, one of the busiest sites on the Web, Wikipedia.org, is an encyclopedia written and edited by volunteers that was built using a content management system called a *wiki*. Wikis started out as a way to make it easy for individuals to quickly publish content on the Web without any knowledge of HTML. Now you can download and use MediaWiki, the tool behind Wikipedia, free of charge. You'll learn more about it later in this lesson.

Content Management in the Cloud

These days, many web publishers do not even host their own content management systems. They subscribe to applications provided by companies that specialize in managing these sorts of applications. So, for example, instead of installing the WordPress blogging tool, you can just sign up for an account at WordPress.com and set up your blog,

avoiding the task of downloading WordPress, installing it on a server, and making sure that it all works. The advantage is ease in getting started and reduced maintenance over time, and the disadvantage is lack of control. When you set up your own copy of WordPress, you can install your own themes and plug-ins, or even modify WordPress itself if you need to. Using the version of WordPress provided by WordPress.com eliminates some of that flexibility.

The other major trend in web development is sites making it easy to use content from websites on other websites. In Lesson 12, "Integrating Multimedia: Sound, Video, and More," you learned how to upload videos to YouTube and then embed those videos in your own pages. Doing so enables you to avoid the hassle of encoding video yourself, renting space to put the videos online, and providing a player that works in every popular browser. Allowing this kind of flexibility is becoming increasingly common. So, rather than going out and finding one tool that gives you all the features you require, you can mix and match tools as needed to offer your users exactly the kind of experience you want.

By combining services that provide content management, user community features, and multimedia hosting, you can offer your users capabilities that, not too long ago, would have been completely out of reach for the individual web publisher. Right now, in an afternoon, you can create a new blog on WordPress.com, populate it with videos hosted on Vimeo, and allow your users to discuss the videos using a third-party comments tool like Disqus (http://disqus.com/).

Is a Content Management System Right for You?

It might seem odd to talk about using an application to manage your content now after you've worked through 21 lessons explaining how to build web pages from scratch. However, those skills will serve you well whether you use a content management system or not. Content management systems can take a lot of drudgery out of web publishing without limiting the amount of creativity you can apply in designing your web pages.

Some work is involved in learning, setting up, and customizing a content management system. When you're creating a website, it's probably easier to start with just a few static files and leave aside the content management system. As your site gets bigger, though, at some point the work involved with dealing with static files offsets the initial investment required to start working with a content management system. You have to figure out whether that initial time investment will be worth the effort.

Another common trade-off with content management systems is that they vary in flexibility. Some, such as photo galleries, are built to handle specific kinds of content. You wouldn't use photo gallery software to manage an online news site, for example. Others are more flexible. For example, Drupal is designed as a generic content management system for any type of content. Generally speaking, the more flexible a content management system is, the more work it is to set up. So, if you know you're publishing a photo gallery, setting up a photo gallery package probably makes more sense than setting up a general-purpose content management system and customizing it to present photos.

In Lesson 20, "Putting Your Site Online," I talked about the various options for web hosting. A similar set of options is available when it comes to content management systems. You can write your own. This provides the most flexibility, but it can be a huge amount of work. You can install and manage an application that someone else wrote and customize it for your own needs. Generally, to go this route, you need your own server or at least an account with a web hosting provider. The final option is to sign up for a hosted application. For many kinds of web applications, free versions are even available. They make sure that the servers stay up and that the application is running properly, and you just focus on entering your content and making whatever customizations are available. Hosted applications are generally the least flexible, but they're also the easiest to get started with.

Types of Content Management Systems

Content management systems can be sorted into rough categories. As you've probably figured out, there's more to it than just deciding to use a content management system. You have to figure out what type of content you have and which content management system makes the most sense for you. In some cases, multiple content management systems may be the answer. You might prefer to publish your writing using a blogging tool and to publish your photos using a photo gallery application. Here's a discussion of some common types of content management systems.

Blogging Tools

It seems you can't escape the term *weblog* (or more commonly, *blog*) these days. A web log is just a website where new items are published in reverse chronological order. The items can be articles, links, recipes, photos, or anything else. Generally the most recent items are displayed on the front page, the site maintains an archive of earlier material, and in many cases the site allows users to publish comments of their own. You can find blogs on just about any topic of interest, and increasingly blogs are integrated into other types of websites. For example, many magazines and newspapers publish blogs as part of

their sites. Companies are using blogs to communicate directly with their customers. Political campaigns are using them to get their message out. In large part, the reason blogs have taken off is that the format is easy to work with. You can publish short items as frequently as you like, and users know that when they visit the site they'll always see the latest and greatest information right up front. In this lesson, I discuss a blogging package you can install yourself called WordPress as well as a hosted blog tool called TypePad. Google also offers a hosted solution that's free to use called Blogger, which you can find at http://www.blogger.com, and many people are starting to use Tumblr (http://tumblr.com), a tool that makes it easy to create a blog around snippets found on other sites. You may want to check them out, too.

Community Publishing Applications

Community publishing applications are similar to blogging tools in that they are usually built around publishing items by date right up front. They differ from blogging tools in that they are more centered around providing features for all the site's users, not just the site's author or authors. Generally, these applications enable users to register as members of the site, submit stories, and engage in discussions among themselves. Sites that incorporate this kind of functionality include http://reddit.com/ and http://digg.com/.

Drupal (http://drupal.org) is the most popular application in this category right now. It's a community publishing system written in PHP. The application is written in a modular fashion so that you can include only the features you want.

Wikis

Wiki-style systems take the most radical and counterintuitive approach to content management. Most wikis not only allow anyone to view articles but also to edit them. Some wikis require users to register before they can view or edit articles, but the wiki philosophy is that every user is an editor. When most people hear about this concept, they imagine that the result will be anarchy, but as long as there is a critical mass of users who want the wiki to be useful, this type of system is resistant to attempted defacement. Wikis generally keep a history of changes made to every article, so it's easy to go back to an older version of a page if someone deletes things he shouldn't or otherwise submits unwanted changes.

The most famous wiki is Wikipedia (http://wikipedia.org), an online encyclopedia written and edited by volunteers. Not only can anyone view the articles in the encyclopedia, but they can also edit them. If you know something about Fargo, North Dakota, that's not already in Wikipedia, you can look it up there and add your own information. Attempted vandalism is taken care of by volunteers who keep an eye out for it, and volunteers handle disputes over controversial content, too.

The name *wiki* comes from the name of the original wiki software, which was called WikiWikiWeb, taken from an airport bus route in Honolulu, Hawaii. To read more about the nature of wikis and their history, why not read about them in Wikipedia? The URL is http://en.wikipedia.org/wiki/Wiki.

Aside that everyone can edit the pages in a wiki, the other distinguishing feature of wikis is that they generally provide their own markup language rather than HTML. Generally wiki markup is less flexible than HTML but also a bit easier to get the hang of. Unfortunately, wiki markup usually differs between wiki software packages, so learning MediaWiki's markup language won't help you out a whole lot if you start using TWiki, for example.

Wikis aren't right for every project. If you want to publish your poetry online, putting it up on a page where anyone can edit it probably isn't a good idea. On the other hand, if you need a site where a group can collaboratively write documentation for a project, a wiki can be a great choice. Here's a list of other popular wiki applications and hosts:

- **WikiWikiWeb (http://c2.com/cgi/wiki)**—The original wiki.
- **MediaWiki (http://www.mediawiki.org/)**—The software behind Wikipedia. I discuss it more later in this lesson.

You can find a comparison of various wiki software packages at http://en.wikipedia.org/wiki/Comparison_of_wiki_software.

There's also a list of hosted wikis that you can use at http://en.wikipedia.org/wiki/Comparison_of_wiki_farms.

Image Galleries

As digital cameras have grown more popular, so too has sharing photos online. There are many photo-sharing sites online that you can use to host your photos. Some popular choices include Flickr (http://flickr.com/), Picasa (http://picasa.google.com/), and SmugMug (http://smugmug.com).

If you just want to upload pictures and share them with your friends and family members, these sites provide an easy way to do that. In some cases, however, you might want to host your own image galleries. A number of popular image gallery applications are available that vary in terms of complexity and the extent to which they can be customized. The good ones take care of creating thumbnails of your images for you so that you can just upload your photos without dealing with image-editing programs.

General-Purpose Content Management Systems

There are far too many general-purpose content management systems to list here. Some content management systems are just toolkits that enable you to build something on your own. Others are highly structured with tons of features that you control via configuration, rather than by writing code on your own. Here are some features that these general-purpose content management systems tend to have in common:

- **Templating system**—All content management systems provide some way for you to specify how your content will be laid out on the page and how the pages containing the content will look. Most templates take the form of HTML files with special markers that indicate where each piece of content goes. In many cases, the templates are just regular PHP, JSP, or ASP pages. In other cases, the content management system will provide its own template language.
- **Data repository**—Content management systems produce web pages, but they usually store the data in some structured format, whether it's in a relational database or in XML files. When users enter data, it is stored so that the system can merge the data with the templates to generate HTML when the documents are published.
- **Workflow system**—Controlling who gets to edit which content as well as managing processes for moving content from draft to published status are big parts of many content management systems. Some provide rigid systems that you have to work within; others let you modify the workflow to suit your own organization. Generally, complex workflow features are part of larger content management systems. They're not useful for individuals or small groups.
- **System for writing and editing content**—All content management systems provide some way for you to get your content into the system. Some enable you to enter it using HTML forms; others provide integration with external tools such as Microsoft Word.
- **System for publishing content**—After content is in the repository and has been approved for publishing, a content management system has to merge that content with a template and publish it on a web page. Some content management systems produce actual HTML files; others generate the HTML on-the-fly for every request. Which approach is used usually depends on how much dynamic content appears on the page. Most content management systems that don't publish static files provide some mechanism for caching so that data that hasn't changed can be read from the cache rather than being generated dynamically.

Working with Packaged Software

So far, this book has been about building things from scratch. This lesson, on the other hand, is about applying tools to make building things a bit easier. However, as I've mentioned, using existing tools can be a job unto itself. Before you can start using MediaWiki to manage your content, you have to download it and get it installed on a server and configured properly. The same is true with any software package. Even if you go with a hosted solution, you still have to set things up so that the software enables you to accomplish your goals.

You may also be wondering at this point why you bothered to learn HTML, CSS, JavaScript, and everything else in between if you're going to let an application do your work for you. This is one of the most common misconceptions about this kind of software. Content management systems make some things more convenient, but you'll still apply all the skills you learned over the course of the book to make your pages look the way you like.

Before discussing specific applications, I'm going to discuss some topics that pertain to nearly all applications of this kind: relational databases and deployment issues.

Relational Databases

In Lesson 21, "Taking Advantage of the Server," and in this lesson, I've brought up the subject of relational databases more than once, but I haven't explained what they are or how they work. Relational databases are by far the most popular data repository for web applications. There are a number of reasons for that:

- **They can scale from the smallest to the largest of applications**—You can start a website on a web hosting account on a server that's shared with 50 other people and eventually wind up running 50 servers of your own and use a relational database to store your data the entire way.
- **They are flexible when it comes to the types of data they can store**—All relational databases store their data in tabular format. Each record is a row in a table, and the columns describe the properties of each record. In other words, if you have a table to store information about people, every person will have the same properties. The actual structure of your tables can be customized to suit just about any task. You can create a table of links that has names, URLs, and descriptions for each link, or a table of invoices that lists the customer, amount, shipping address, and item ordered for each invoice. The columns in the table can be configured to hold numbers, dates, short strings, large bodies of text, or even binary files.

- **You can create relationships between tables**—That's where the "relational" in relational database comes from. You can relate articles to authors, invoices to customers, or even people to other people within the same table. The relational model is very, very powerful.
- **Relational databases use a standard query language called SQL (Structured Query Language)**—Although some differences exist between database vendors, if you know SQL you can work with just about any relational database.
- **Relational databases are popular**—The basic technology behind relational databases hasn't changed much in the past 20 years, and they are the foundation of many business applications. What has changed is that now several relational databases are freely available, providing powerful features once only associated with the most expensive software targeted at large corporations. Relational databases are so popular and common that using them for data storage for an application is often an easy decision.

Relational databases are usually deployed as a service that is accessible over the network, much like an HTTP server. The difference is that instead of using HTTP to communicate, the database will use its own proprietary protocol. To communicate with a database, you generally need some kind of client or library provided by the database vendor. PHP has clients for most popular relational databases built in, so you can just use the functions associated with your database and not install anything extra.

Most hosting providers that allow you to use PHP, CGI scripts, or some other scripting environment also provide access to relational databases. You may have to fill out a special form to have a database set up for you. After it's set up, your hosting provider will give you the hostname of your database server and a username or password. You can use that information to access the database, or in the case of packaged software, you can use it to configure the software so that it can access the database.

After a relational database is installed, all the administrative tasks—from creating specific databases to creating and managing users—are accomplished using SQL. So to configure a database for a particular application, you generally create a new user and database for it to use and then run an installation script that creates all the tables that the application needs and inserts the initial data into the database. You hosting provider will often take care of creating the user and database, and you will run the installation script that builds the initial database.

The most common relational database offered by web hosting providers is MySQL, an open source database that's freely available. There are many other popular relational databases, too, including Oracle, Microsoft SQL Server, and PostgreSQL (another free database). For web applications, however, MySQL is the leader. All the applications I'm going to discuss in particular work with MySQL. If you want to read more about it, the manual is available online at http://dev.mysql.com/doc/.

Deploying Applications

All the applications discussed in this lesson run on a web server. When it comes to hosted applications, you don't need to worry about the web server. That's the job of the application provider. In cases where you are installing your own applications, it's up to you to make sure your web server environment will work with your application. For example, all the applications I'm going to discuss in detail are written in PHP. If you are planning on deploying to a server running Microsoft Internet Information Services without PHP support, you won't get these applications to run. Before you download and attempt to install an application, check its requirements to make sure the server on which you plan to deploy the application meets them.

If you lease server space from a web hosting provider, you may need to check with it before installing your software to make sure it's compatible with their environment. For example, in addition to big requirements such as PHP or MySQL, some software packages also require particular libraries or supplemental software packages to function properly. You might need to ask to find out whether your hosting provider has all the software you need running in its environment. In some cases, a hosting provider will actually install the software you need if you explain what you're trying to do.

There's another issue you might run into depending on what kind of access you have to your server. Generally, two levels of access are available when it comes to web providers: FTP access and shell access. If you have FTP access, you're allowed to upload your files to the server, but you are not allowed to log in to the server and run commands directly. Shell access provides a command-line interface to the server that enables you to edit files directly on the server, move things around, and generally use the server more like it's your own computer. If you don't have shell access to the server, you will have to edit the configuration files for your application locally on your own computer and then upload the files and test them on the server.

TypePad: A Hosted Blogging Application

As I've said before, the shortest route to using a content management system is to sign up for a hosted application. You don't have to download the application, deal with a hosting provider, run installation scripts, or tweak configuration files. TypePad is a subscription service for publishing a blog. It provides all the features you'd expect from a blog—archives, user comments, and syndication feeds. You can use it without supplying your own domain name, as long as you're willing to accept a name that ends with "typepad.com," and you don't even have to modify the templates if one of the default themes is okay with you.

As soon as you've signed up for TypePad, you can start posting articles to your new blog. Figure 22.1 shows a screenshot of the TypePad posting interface. It's fairly typical of the posting interfaces in most content management systems. For blogs, the typical properties of an article are title, category, and the article text. You can enter all those values in this form. TypePad also includes some basic workflow features. You can publish the post immediately, save it as a draft, or schedule it to be published at a specific time.

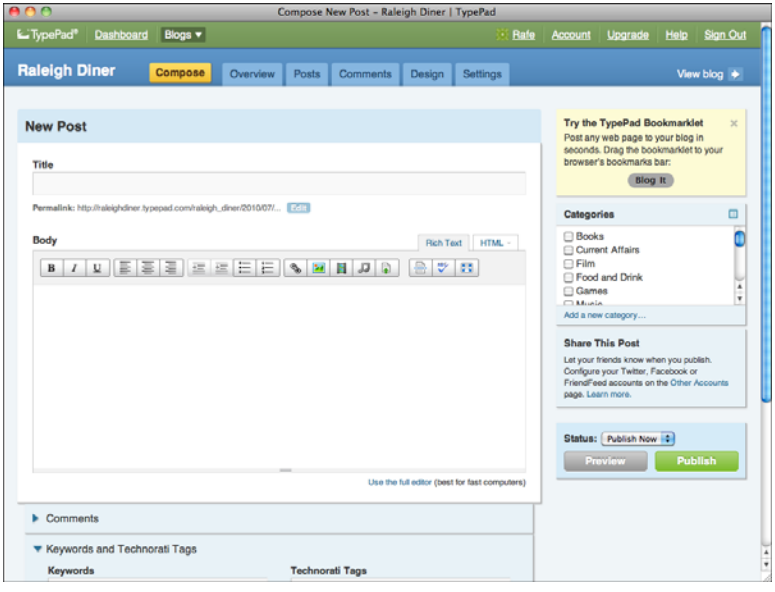


FIGURE 22.1
The TypePad publishing interface.

To write a post and publish it, you just have to fill in the fields and click the Save button. At that point, your post will appear on the public web page for your blog, as shown in Figure 22.2.

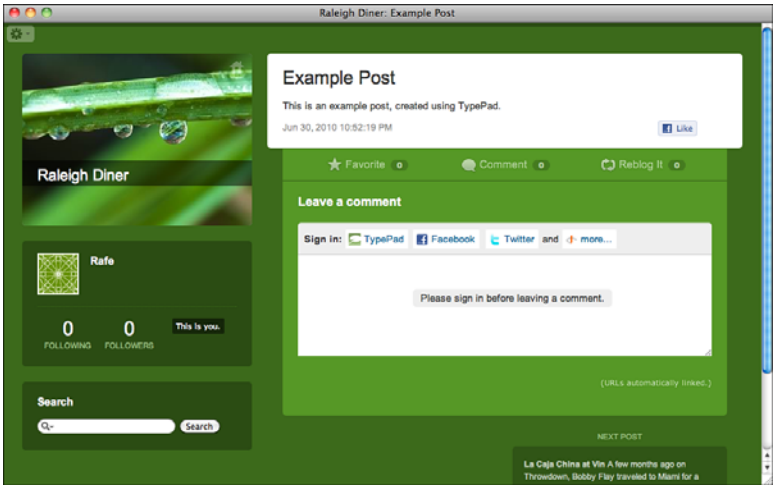


FIGURE 22.2
A public post on a weblog.

Even for a relatively basic service such as TypePad, there are still a lot of configuration options. Figure 22.3 shows TypePad's blog configuration page. As you can see from the figure, TypePad's Control Panel has several tabs, all of which reveal various configuration options. This is one difference between hosted applications and applications you install and maintain. Generally, applications that you install include configuration files that you must edit, along with configuration forms like the ones found in TypePad.

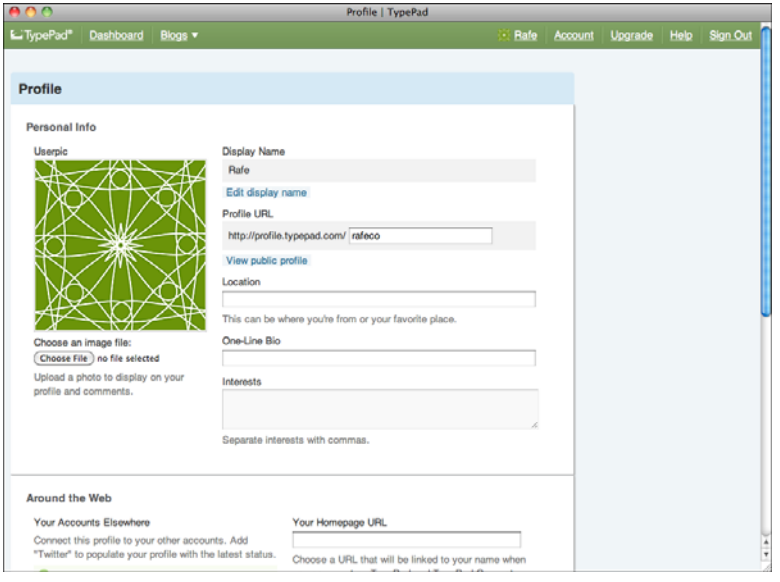


FIGURE 22.3
The TypePad author profile configuration page.

One more thing to discuss regarding TypePad is how to modify the design for your pages. TypePad allows you to change both the theme and the layout for your pages. Themes alter the color scheme and visual appearance of your pages. Layouts dictate where the various components of the page are placed, such as the blog entries, links to the archive, photo galleries, and other pieces of content you publish. TypePad includes a set of themes you can choose from, as shown in Figure 22.4.

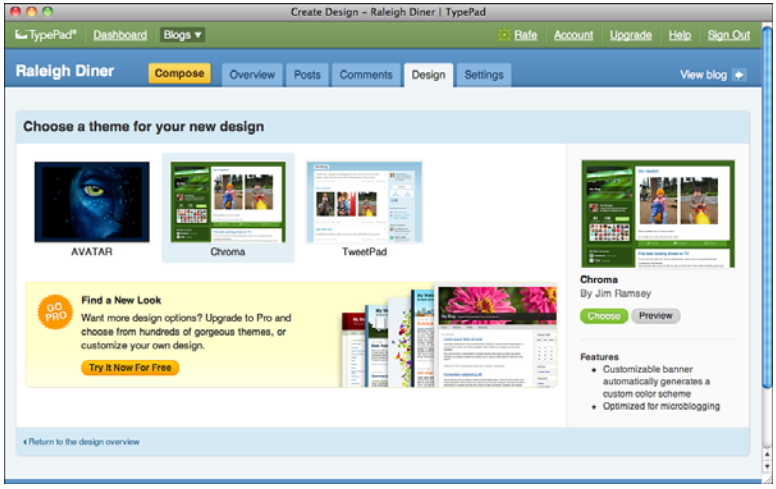


FIGURE 22.4
Selecting a theme in TypePad.

If you prefer to design something, TypePad provides that option, too. The degree of flexibility you have in modifying your design depends on which level of TypePad account you sign up for. Users with Pro accounts can edit the style sheet for their blog directly.

There are several other subscription (and free) blog services, too, including WordPress.com (http://www.wordpress.com), Tumblr (http://www.tumblr.com), and Blogger (http://www.blogger.com). They all offer feature sets similar to TypePad.

WordPress

WordPress is a blog publishing tool written in PHP. It's open source software, so you are free to use it or modify it as you want. You can download WordPress at http://wordpress.org/. The first step is to extract the file in the place where you want to install your blog. The files in the archive are all inside a folder called `wordpress`, but you can move them into another directory if you like.

TIP

Many Web hosting providers offer one-click installation of WordPress and other popular content management systems from their control panels. If you want to use WordPress, or any of the applications discussed in this lesson, you should check to see if your hosting provider offers this feature.

After you've extracted the archive, you should copy the `wordpress` directory to a directory accessible as part of your website. If you want your blog to reside at the root directory for your server, copy the files out of the `wordpress` folder and directly into your document root directory. Visiting the home page of your WordPress installation will quickly illustrate the difference between using a hosted service and running an application on your own. You're greeted with a request to create a configuration file, as shown in Figure 22.5.

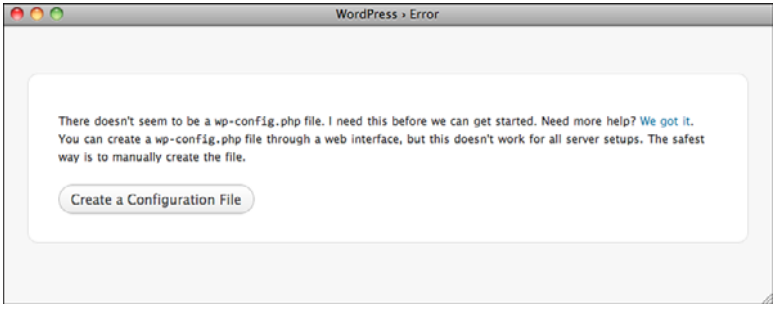


FIGURE 22.5
WordPress needs a configuration file to get started.

In your WordPress directory, you'll find a file named `wp-config-sample.php`. To set up WordPress, you need to copy this file to a file named `wp-config.php` and enter your database settings. If you are deploying WordPress to a server where you don't have a shell account and can't edit files directly, you must create and edit this file before you upload your files to the server.

I'm installing WordPress on my own computer, so I created the database and database user myself. If you are installing WordPress on a shared hosting account, chances are you've gotten the database login information from your hosting provider. In some cases, you may want to install several different applications, or perhaps even multiple installations of WordPress in the same database. In that case, you'll want to look at the following line in `wp-config.php`:

```
$table_prefix  = 'wp_';   // Only numbers, letters, and underscores please!
```

This is the string that WordPress puts on the front of the names of all the files it creates. If you already have one WordPress installation using a database, you'll want to change this prefix to something else so as to prevent the two installs from conflicting. I have created a database just for this purpose, so I'll leave the prefix as is.

After you have `wp-config.php` in place, you can proceed with the installation. If you reload the index page in the WordPress directory, you'll be referred to `install.php`, unless there's a problem with your database configuration. If WordPress can't connect to the database using the settings you provided, it will print an error message. Make sure you have all the information on your database entered properly and test your database connection using this page until WordPress stops complaining.

In the first step of the actual installation, shown in Figure 22.6, you pick a name for your blog, create an administrative user, and enter your email address. After you've done that, WordPress creates all the database tables it needs and generates an account for you to log into the system. That's it for installation.

FIGURE 22.6
The WordPress installation page.

After you've completed the installation, you're presented with the Dashboard for your blog. It's not all that different from the Control Panel for TypePad. The posting page, shown in Figure 22.7, is even more similar to TypePad and most other blogging tools. You can enter the title and body of your post, along with some other optional information. The options on the right side enable you to mark your post as draft or published as well as categorize it. You can select from other options, too.

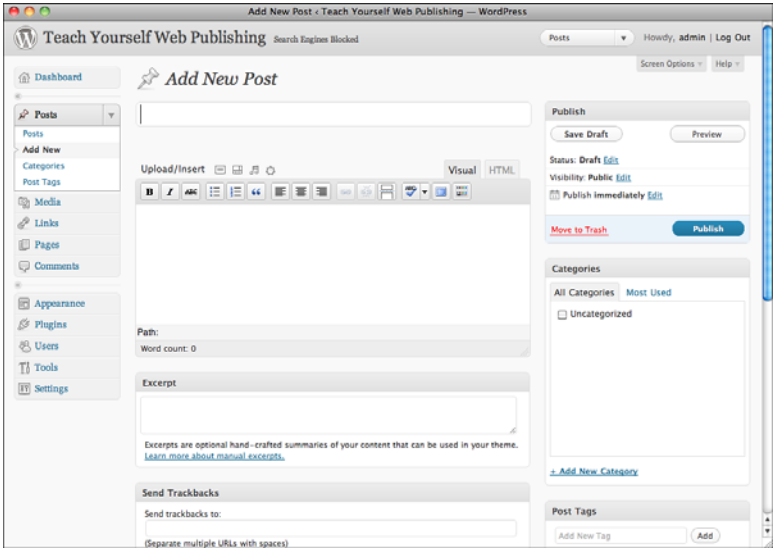


FIGURE 22.7
The posting page for WordPress.

The posting page isn't all that interesting. Where WordPress differs from TypePad is in how much control you have over the layout and design of your blog. With WordPress, all the work you've done learning HTML, CSS, and PHP can pay off. Clicking the Appearance menu item and then clicking Editor enables you to edit all the files that make up the published side of your blog from within your browser (see Figure 22.8).

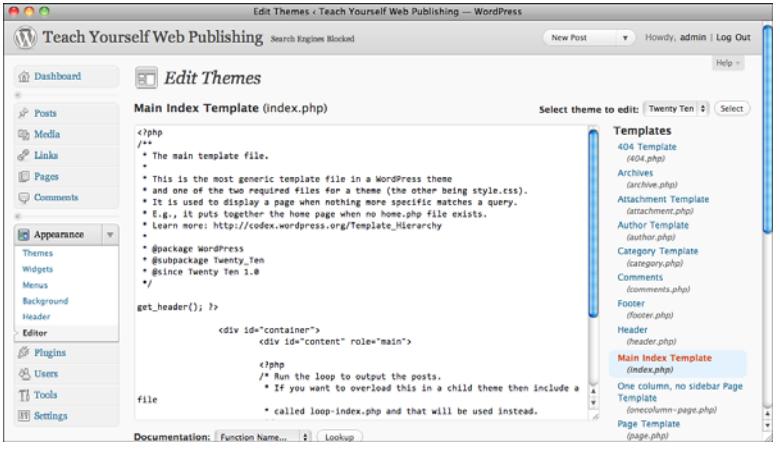


FIGURE 22.8
Editing part of a WordPress theme.

You can see the actual source code for `index.php`, the main page of my WordPress site, in a `<textarea>` element. Any changes I make to the file will be saved to disk, and the next time I view the front page of my blog, I'll see them.

If you know how to edit these files, you can customize WordPress to your heart's content. If you don't want to make changes to your WordPress theme by hand, you can download and install themes that other people have created at http://wordpress.org/extend/themes/. You can download the themes and place them within your WordPress installation so that they can be applied to your weblog. You can also use those themes as starting points to create your own custom theme.

The functionality of WordPress can also be extended using plug-ins. This is yet another advantage of running your own installation of WordPress. Not only can you create your own themes and install themes created by other people, but you can also extend WordPress using plug-ins that you download or you can write the new functionality yourself. A directory of available plug-ins can be found at http://wp-plug-ins.net/. This directory contains hundreds of plug-ins for WordPress.

Installing plug-ins is straightforward. Let's say I want to install Yet Another Related Posts plug-in, which automatically lists posts related to the post currently being viewed in WordPress. First, I go to the download site and download the plug-in. Here is the URL for the plug-in:

http://wordpress.org/extend/plug-ins/yet-another-related-posts-plug-in/

After I've downloaded the file, I have to expand the archive and copy the plug-in directory into the `wp-content/plug-ins` directory in the WordPress installation directory. At that point, I can click the Plug-ins tab inside WordPress and I should see that the Related Posts plug-in is installed, as shown in Figure 22.9.

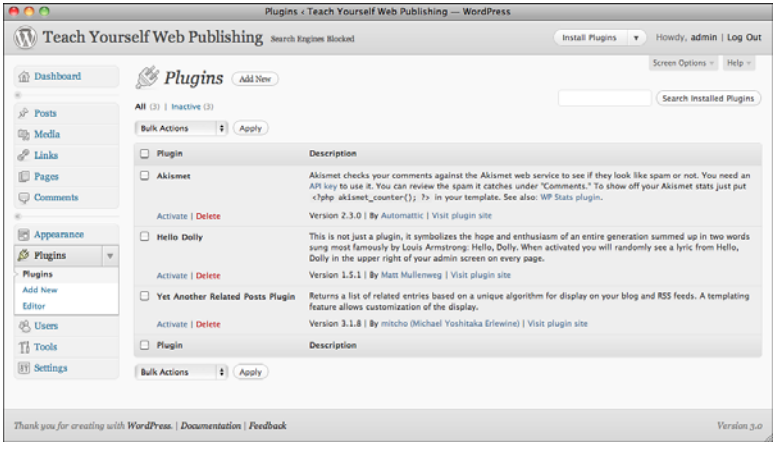


FIGURE 22.9
The Related Posts entry on the WordPress Plug-ins page.

After the plug-in is installed and activated using the Activate button on the Plug-in list, I can configure the options for the plug-in using the Related Posts Options tab that appears

once the Related Posts plug-in is activated. Then all I have to do is go to my single-page archive post and find a spot for the `related_posts()` call to include the list of related posts.

The biggest advantage of installing your own copy of WordPress is the vast number of themes and plug-ins that are available. If you want to use WordPress but you don't want to deal with the overhead of setting it up, you can create your own WordPress blog at http://wordpress.com/.

MediaWiki

Many applications are available for publishing wikis. I've chosen to discuss MediaWiki because it's written in PHP, so it can be installed at a wide range of web hosts, and because it is relatively simple to set up. MediaWiki is also widely adopted, and it's the plumbing for Wikipedia, the most popular wiki. Because of its popularity, the software is undergoing constant improvement, and many extensions are available for the software.

The main downside of publishing a site using MediaWiki is that it won't give you a great opportunity to use or improve your HTML skills. Content in MediaWiki is entered using MediaWiki's internal markup language. MediaWiki markup is translated into HTML tags, though, so even though you use different syntax, knowledge of how web pages are constructed will still help you lay out your pages using MediaWiki's markup language. Here's an example of MediaWiki markup:

```
== Sample Heading ==

This is a paragraph.

This is a paragraph containing '''Bold text'''.

Here's an [http://www.example.com outbound link].
```

And here's the equivalent in HTML:

```
<h2> Sample Heading </h2>
<p>This is a paragraph.</p>
<p>This is a paragraph containing <b>Bold text</b>.</p>
<p>Here's an <a href="http://www.example.com">outbound link</a>.
```

The first thing you should notice is that unlike HTML, in MediaWiki markup, white-space counts. When you skip lines between paragraphs, MediaWiki converts those breaks into paragraph tags. Headings are specified using equals signs. Two equals signs are converted to `<h2>`, four are converted to `<h4>`, and so on. MediaWiki markup can also be

used to apply inline tags as well as block-level tags. As you can see, I used `'''` to make some text bold, and I created an outbound link from the wiki. You can also use MediaWiki markup to create tables, include images in pages, and take on most of the other tasks that you can accomplish with HTML. There's a guide to MediaWiki's markup language at the following page:

http://meta.wikimedia.org/wiki/Help:Editing

This is a wiki page, and if you had some information to add to the discussion of wiki markup, you could add it.

Downloading and Installing MediaWiki

The download and installation process for MediaWiki is similar to the process for WordPress. To download the software, go to http://www.mediawiki.org/wiki/Download.

Find the most recent release and download that archive. After you've expanded the archive, rename the folder, which probably has a name such as `mediawiki-1.15.4`. The name you give it will appear in the URL, so you probably want to pick `mediawiki` or just `wiki`. Upload or copy all the files into the document root of your server (or into the directory where you want the wiki to reside), and then go to the URL where you just uploaded the files. You'll see a page like the one in Figure 22.10, which indicates that you need to go through the MediaWiki installation process.

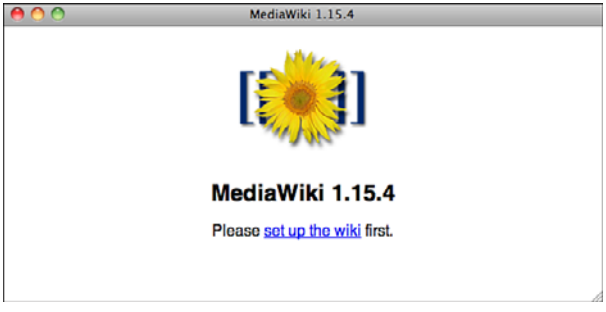


FIGURE 22.10
MediaWiki needs to be installed.

When you click the Setup link, the MediaWiki install script checks to make sure your server is configured properly and has the software that you need for MediaWiki to work. It also asks you to answer a few questions. Some are easy, such as the name of the site and the email address for the person running the site. Others are not so easy, such as the caching configuration. You can safely choose No Caching for that one.

If MediaWiki is able to write to your configuration files (based on the permissions on those files and the privileges that the web server runs with), it allows you to enter all

your database configuration settings right on the installation page. If you have root (administrator) access to your MySQL server, MediaWiki will even create a dedicated database and user for you. If you're using shared hosting, you'll probably have to enter the database name and login information for your database and then let MediaWiki create its tables. MediaWiki, like WordPress, lets you pick a prefix for your table names so that you can avoid naming conflicts if multiple applications use the same database. After setup is complete, MediaWiki requests that you move the configuration file it created to the proper location, and then you can begin editing content.

If everything is set up properly, MediaWiki installation can be painless. Unfortunately, with these types of applications, there's always an opportunity for something to go wrong. If you run into trouble, check out the MediaWiki installation guide:

http://www.mediawiki.org/wiki/Manual:Installation

The settings for MediaWiki are found in the file `LocalSettings.php`, which MediaWiki may have generated for you. You can find a full list of configuration settings at the following location:

http://www.mediawiki.org/wiki/Help:Configuration_settings

MediaWiki will work fine with the default settings, but you'll probably at least want to add your own logo to replace the placeholder image. MediaWiki also supports themes, which are called *skins* in the MediaWiki world. A big list of skins that you can download and install can be found here:

http://www.mediawiki.org/wiki/Gallery_of_user_styles

Users can change their personal settings to use any skin that's installed. To change the default skin for the site, edit the `$wgDefaultSkin` variable in your `LocalSettings.php` file.

Using MediaWiki

After you have MediaWiki set up however you like, you can start entering content of your own. If you go with the default installation, anyone who sees your site can start entering content of her own. Every page in a wiki has an edit link on it, allowing you to jump in and make changes. To add an internal link in MediaWiki, you enclose the name of the page in double square brackets, like this:

```
[[My New Page]]
```

When you click the link, you'll be asked to fill in the body of the new page. That's all there is to it. You can also create a new page by entering its name directly into the URL. If you enter a URL, such as http://localhost:8888/mediawiki/index.php/Flying_Monkeys, and a page named "Flying Monkeys" does not exist, it will be created automatically.

Wikis tend to grow organically; as people need new pages, they create them by linking to them. With a wiki, it's easy to move content from one page to another, split one long page into several smaller pages, and generally manage your content however you see fit.

MediaWiki keeps track of every change made to every page. On each page, you see a History tab that you can click on to see all the edits ever made to that page. The history page for the sandbox page on the MediaWiki site appears in Figure 22.11.

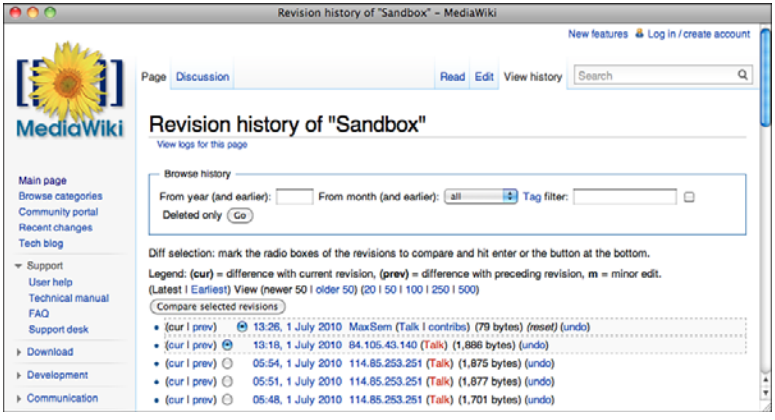


FIGURE 22.11
The edit history for a wiki page.

From the History page, you can compare revisions of a page or view the old revisions. To revert a page back to the old version, view that version, click Edit, and then save the page. MediaWiki will warn you that you're replacing newer content, but if you're reverting changes, you can ignore the warnings.

Drupal

Drupal is a general-purpose content management system that continues to gain popularity. Out of the box, it is set up to create a blog-like site that allows authors to publish news and allows the public to log in and comment on the stories. Unlike applications like WordPress and TypePad that are strongly focused on publishing blogs, Drupal supports a wide variety of applications.

Drupal provides flexibility in three main ways. The first is that a large numbers of modules are available to provide new functionality. There are Drupal modules that allow you to manage members in an organization, set up and operate an online store, or integrate with other sites. The second is that the functionality of the Drupal core and most of the modules can be customized through a web interface without any programming required.

So, if you want to add a pet registry to a Drupal site, you could add a new content type for pets with its own custom fields without modifying the database on your own or modifying the Drupal source code at all. And finally, like WordPress, Drupal supports themes, enabling you to customize the look and feel of a Drupal site.

The installation and setup of Drupal is similar to the process for WordPress and MediaWiki. It's also a PHP application that uses MySQL as its database. Setting it up as is as simple as updating the configuration file, uploading the application, and then performing the steps advised in the web interface.

Instead of walking you through the precise steps to install yet another application, I'll go straight into talking about some of the things you can do with Drupal. If you do want to install it, you can check out the installation guide at http://drupal.org/getting-started/install. If you prefer, you can set up a hosted Drupal site at Drupal Gardens (http://drupalgardens.com/). There's an advertising-supported version of Drupal available there that you can use for free, and there are ad-free options to which you can subscribe.

Using Drupal

The easiest way to get started with Drupal is to sign up for a free account with Drupal Gardens. After you've registered for a new account, you just have to choose a URL for your site, and then you can start creating the site. Drupal Gardens starts by asking you which features you're going to want to set up out of the box, as shown in Figure 22.12.

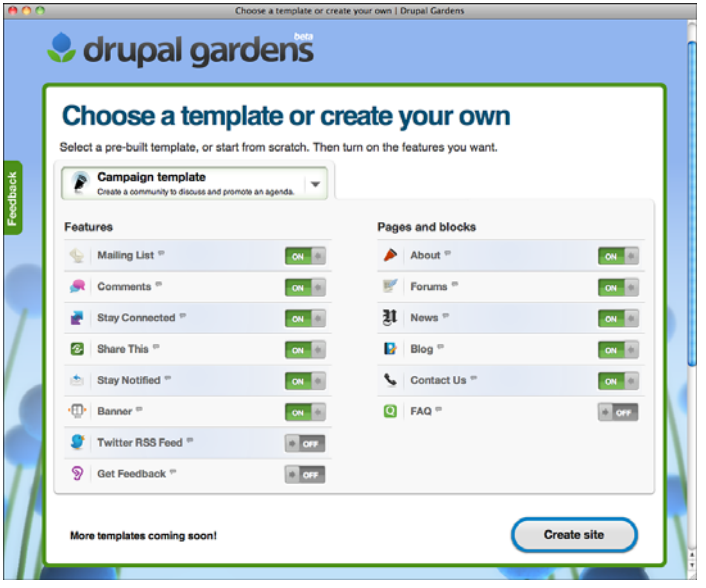


FIGURE 22.12 Creating your Drupal site.

After you've set up your site, you can begin to customize its look and feel and start entering content. Drupal's administrative interface shows the page as it will look to end users, along with some menus and controls that are visible only to administrators. As you can see in Figure 22.13, the main area of the page features the default template and some placeholder content, such as the default slogan and site name. The gray and black navigation bars across the top of the screen are the administrative controls. So, for example, to update your site name and slogan, you click the Configuration button and then click Site Information in the Configuration dialog box.

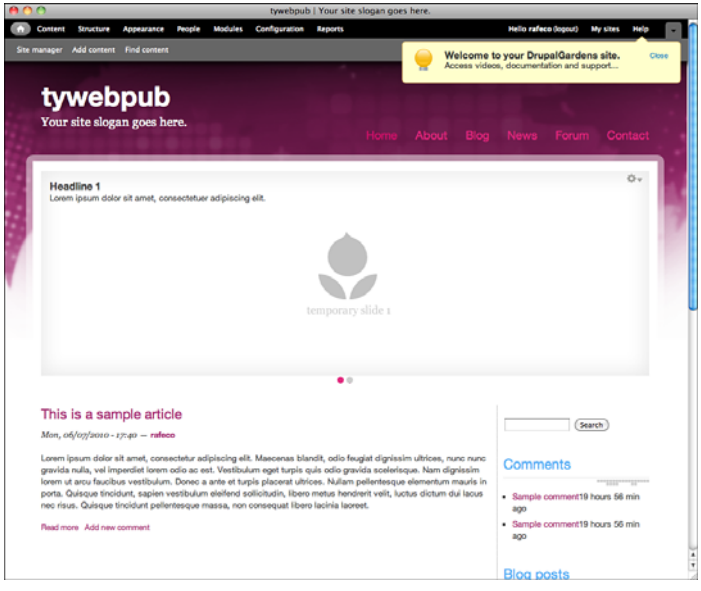


FIGURE 22.13
The home page of a new Drupal site.

Drupal supports two kinds of content. There are blocks, which are reusable chunks of content that are used throughout the site. Blocks include things such as navigation menus, search forms, or lists of recently posted content. Drupal themes are split up into regions, and administrators can place blocks in whatever region they like. Clicking the Structure button enables you to create blocks and assign them to various regions on the page. New Drupal sites already provide a number of commonly used blocks, arranged into appropriate regions in the layout. They can all be rearranged through the administrative interface.

Nodes are the second type of content. Unlike blocks, which are reused throughout the site, nodes are individual bits of content that may be found in lists that are displayed in blocks, and which have their own detail pages. Nodes include things such as web pages, articles, blog entries, forum posts, and photos. One of the big parts of setting up your

Drupal site is deciding which types of nodes will be part of it. Nodes are the "content" in the content management system.

To create new nodes or manage the nodes that already exist, you can click the Content button in the navigation. When you do so, the Content dialog, shown in Figure 22.14, will be displayed.

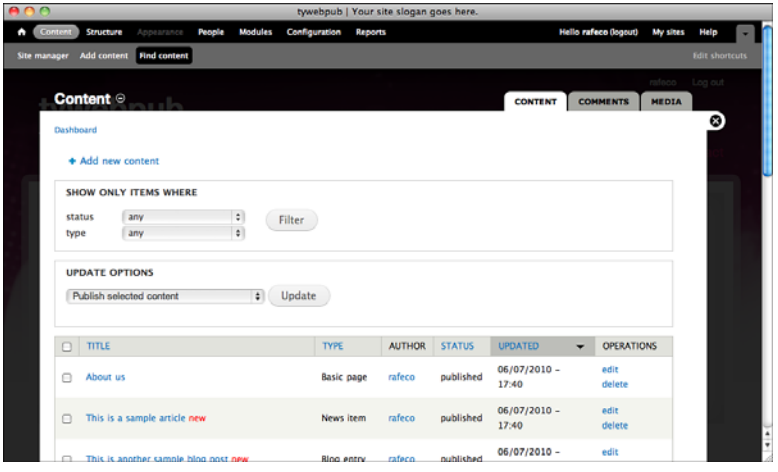


FIGURE 22.14
The Drupal Content dialog.

As you can see, the dialog provides a list of existing content, enabling the administrator to add new content or manage the content that already exists. When you click the Add New Content link, you're given the option of Drupal provides a WYSIWYG editor written in JavaScript for editing content, and also provides the option of editing the HTML source for the content, too.

To change the appearance of your site, you can select a new theme, and then modify by choosing an alternate color scheme or even editing the styles of individual elements on the page. You can see the Theme selection interface in Figure 22.15.

After you've selected a theme, you can further customize the look and feel of your site using the Brand, Layout, and Styles buttons. The Advanced button enables you to add custom Cascading Style Sheets (CSS) to your site if you need to further customize your site.

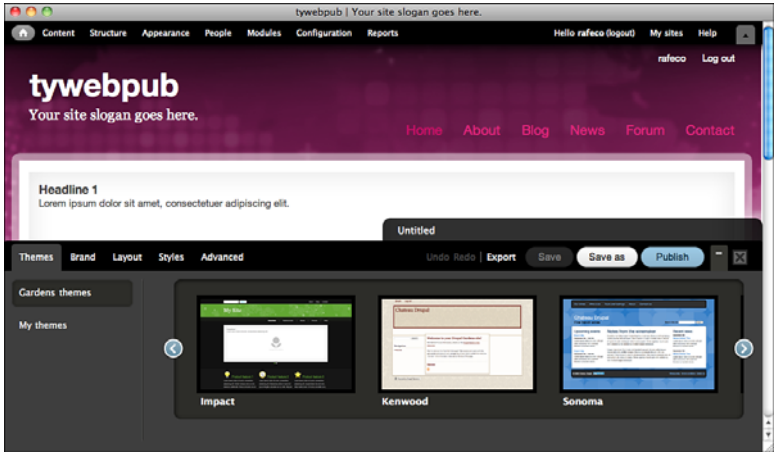


FIGURE 22.15
The Drupal Garden theme selection interface.

Finally, you can change the functionality of your site by enabling and disabling modules. To do so, you click the Modules button in the top navigation bar, which will reveal the Module configuration page, shown in Figure 22.16. If you want to remove the forum functionality from your site, you just go to the Module configuration page and disable the Forum module. If you want to enable Google Analytics for your site, you can enable the Google Analytics module. The ability to control the functionality of your site by enabling and disabling modules is one of the key advantages of Drupal. If you set up your own copy of Drupal, you can also download and install modules on your own.

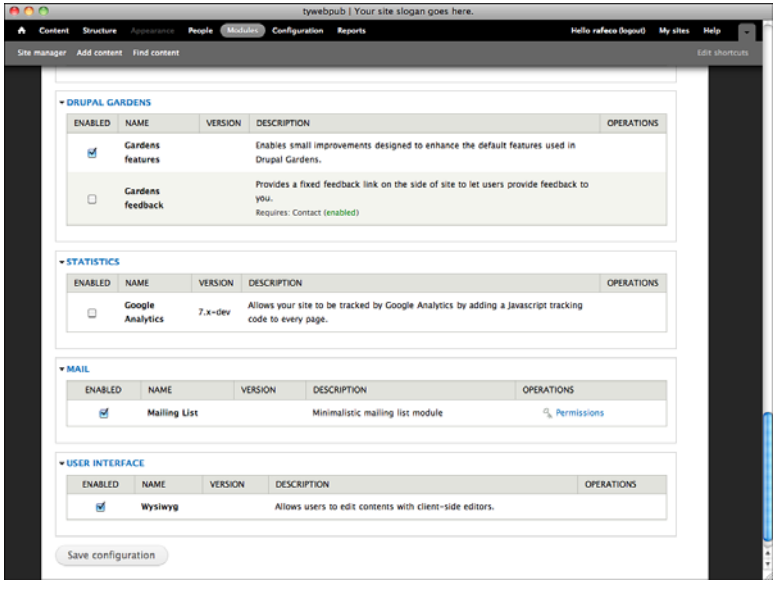


FIGURE 22.16
The Drupal Garden theme-selection interface.

As you can see from the screenshots, there's a lot more to Drupal than the few simple concepts I introduced. The popularity of Drupal is growing because it provides the flexibility to create many kinds of sites without requiring programming skills on the part of the programmer. The downside is that Drupal is rather complex. If you just want to publish a weblog or a few static pages, applications like TypePad and WordPress are easier to get started with. However, if your site will eventually grow to encompass many kinds of features, you may find that Drupal better suits your requirements.

Incorporating Dynamic Content from Other Sites into Your Pages

One alternative to content management systems is to create your content in a hosted web application and then use widgets provided by that application to integrate the content into your own web pages. You've already learned how to embed videos hosted by YouTube into web pages. It turns out that many other sites provide the same functionality. The modern Web is all about integrating sites to take advantage of the strengths of each.

It's common to set up a blog or website using a tool like WordPress and then to augment the capabilities of that software by incorporating content from other sites, too. So, you can use your content management system for what it's best at, and you can use other sites for what they're best at, and incorporate it all into a single website. In this section, I talk about widgets because they're easy to use. You go to a website, enter the information you need to create the widget, and then paste the embed code into your page, just as you would with a YouTube video.

These sites also provide ways to integrate their content into your site that you can take advantage of with your favorite server-side programming language, but such techniques are beyond the scope of this book. You may want to look into plug-ins for your content management system that can be used to pull content from the sites that you use into your own site.

Using Photos from Flickr

Flickr (http://flickr.com) is a popular photo hosting site owned by Yahoo! Flickr is a photo sharing community. You can upload your own photos, view and comment on other people's photos, and create groups and photo pools. You can also use photos on Flickr on your own site. Many Flickr users license their photos under terms that allow them to be used on other websites. If you want to use a photo of a willow tree to post on your own site, you can find a properly licensed photo on Flickr to use rather than taking one yourself. To find photos, go to http://www.flickr.com/creativecommons/, shown in Figure 22.17.

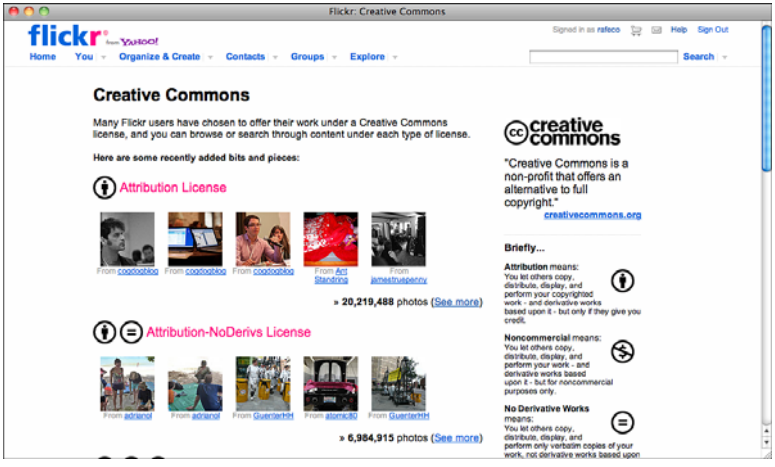


FIGURE 22.17
The Flickr Creative Commons page.

All the photos available through that page are licensed under a Creative Commons licenses. These licenses are defined on the page, but in short, photos shared under these licenses are legal to share, as long as you follow the terms in the license. For example, a photo shared under a Noncommercial Attribution license can be used but not to make money and also requires that the person using the photo credit the person who owns it when it's shared. You just need to find the license that is compatible with your project in the list and click See More.

From there, you can use the search form to search for images that satisfy your requirements. From there, you can download the images and then use them on your own site. Just go to the detail page, click All Sizes in the Actions menu above the photo, and you'll be taken to the page that shows all the image sizes available, shown in Figure 22.18.

NOTE

Flickr's Community Guidelines state that you must link back to the Flickr page for a photograph when you use that photograph on another site. If the photo uses a Creative Commons license that may not necessarily be the case, but it's still polite to do so.

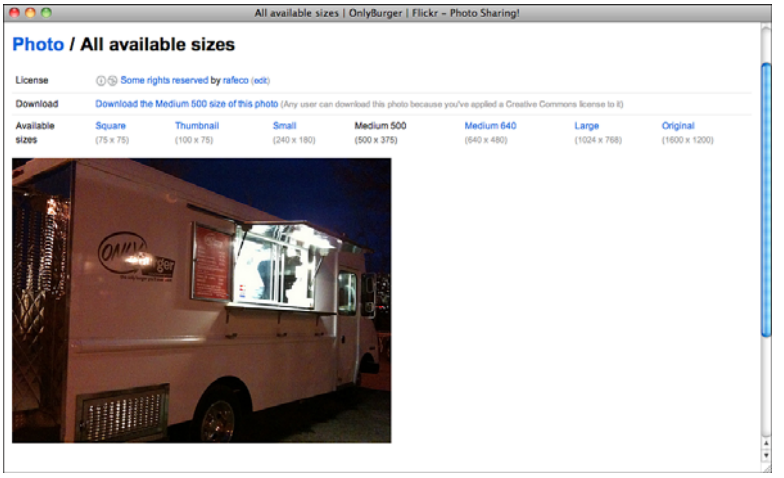


FIGURE 22.18
The Flickr All Sizes page for a photo I uploaded.

The All Sizes pages for photos you upload include the code you need to use the photographs on your own site. If you publish a blog, or a regular website with a blog-based content management system, you can also set up Flickr so that photo detail pages include a Blog This button that makes it easy to use photos on Flickr on your site. To do so, go to Your Account, then click the Sharing & Extending tab, and then add them in the Your Blogs section.

Embedding Twitter Content

Twitter is a social site that enables users to posts messages less than 140 characters long. When you sign up for a Twitter account, you can sign up to follow other people and a feed will be created containing all the posts that they write. Likewise, people can sign up to follow your posts. Not only is Twitter widely used by people to keep track of what their friends are up to, but it's also used to by businesses to post special offers and information about their products, and by publications and blogs to let people know what they're publishing on their sites. For example, my favorite ice cream place posts their flavor of the day on Twitter every day at around noon.

You can follow Twitter using the website at http://twitter.com/, or you can download special applications that are especially designed to work with the site. As a web publisher, you can also incorporate content from Twitter into your own site. For more information, go to http://twitter.com/goodies/widgets. It provides a website that will publish tweets (that's the name for individual posts on Twitter) from a single user, from a list of users, or search results for a particular term. Let's look at how to create a Twitter widget for one user's tweets.

The widget for a single user is called the Profile Widget; to create the widget; go to http://twitter.com/goodies/widget_profile and then enter the name of the Twitter user whose tweets will appear in the widget. Twitter will then provide a preview of the widget, shown in Figure 22.19, along with a button that generates the HTML to embed the widget in your own web page.

FIGURE 22.19
Preview of the Twitter Profile widget.

You can then take the code that is generated and paste it into your own web page to incorporate the widget into your site.

Integrating with Facebook

Like Twitter, Facebook provides a number of features for developers that enable you to integrate its features into your own site. Whereas Twitter's approach is to allow you to use content from Twitter on your site, Facebook's approach is about letting you integrate your site into Facebook. However, Facebook has many millions of users, and integration with Facebook can be a compelling feature for your users.

One of the simplest ways to integrate Facebook with your site is to add a Like button. When Facebook users click the button, the page that they're on will be shared in their activity feed on Facebook. So, if you write an article on the best way to grill a hamburger,

and Facebook users click the Like button, all of their friends will see that they liked it and may click the link, too. This can be an easy and powerful means of promoting your site.

The good news is that adding Facebook's Like button is straightforward. To generate the code to include it on your site, go to http://developers.facebook.com/docs/reference/plug-ins/like. Just fill out the form to get the code you need to add to your pages. When you paste that code into your page, a Like button will appear that your users can click to share your site on Facebook.

You can also create a widget that tracks all the mentions of your website on Facebook. To do so, you use the Activity Feed plug, which can be found at http://developers.facebook.com/docs/reference/plug-ins/activity. As with the other widgets, you fill out the form on the page, and Facebook provides you with the code that you can paste into your own page. An example of the widget along with the form used to create it is in Figure 22.20.

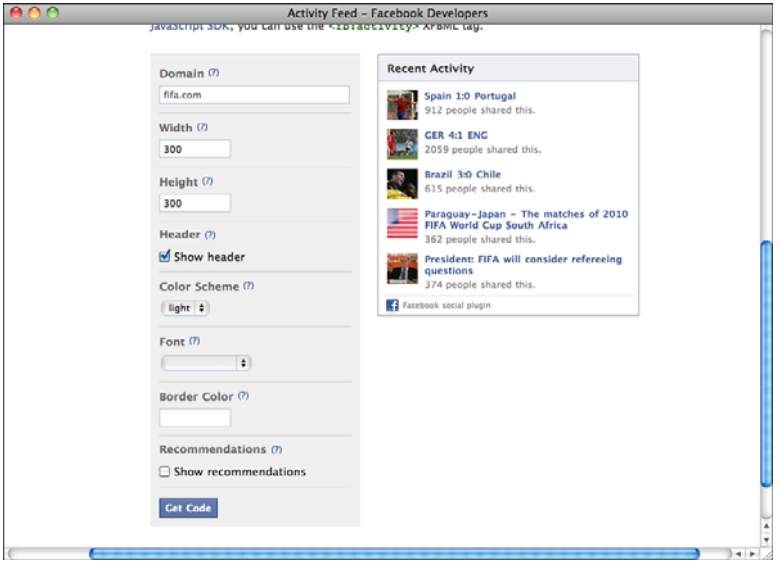


FIGURE 22.20
Preview of the Facebook Activity widget.

Facebook provides a number of other widgets, too. To get an idea of all the ways you can integrate your website with Facebook, go to http://developers.facebook.com/.

Other Applications

This lesson provided a brief tour of a few popular web applications that you can use to go beyond building websites made up of static HTML pages. Not only are there lots of other applications out there, but there are also lots of other categories of applications out there. You can find guest books, shared calendars, discussion boards, tools for sharing bookmarks, and everything in between. Also, even though I focused on applications written in PHP, there are equivalents of these applications written for many other platforms, too.

Spam

Applications such as blogging tools and wikis enable you to build a relationship with your users by allowing them to contribute to your site, either by posting comments or contributing information of their own. Unfortunately, spammers looking to advertise their websites or raise their search engine rank also take advantage of these features by way of programs that seek out and automatically post to sites running common software, such as TypePad, WordPress, and MediaWiki (among others).

There are a number of approaches to combating spam. These days, most popular applications provide tools that attempt to prevent spammers from posting; but even so, some spam still makes it through. I mention this only to let you know that it's a risk of deploying these kinds of applications on the Web these days.

The most important thing to remember is that you shouldn't deploy one of these applications and then abandon it. A MediaWiki installation that goes unused will be overrun with spammers in no time. The same is true for the comments sections of blogs, too. If you put up an application but then stop using it, you should remove the files or configure your web server so that it is no longer publicly accessible. Not only will it keep spammers from filling up your databases with junk, but it will also be an act of good citizenship on the Web.

Summary

The purpose of this lesson was to take you beyond the realm of HTML, CSS, and JavaScript and into the world of web publishing. These days, there's more to publishing on the Web than uploading HTML files to a server. Using free or inexpensive tools, you can build interactive websites that help you keep a handle on your content and make it easy to communicate with your users. You can apply your newfound skills to personalizing and improving these tools as well as to creating the content that you use them to publish.

Workshop

This final workshop contains some questions about content management systems, as well as a quiz and exercises.

Q&A

Q What about security? I've read about security holes in some web applications.

A A number of popular web applications have been found to have bugs that can cause security problems, and when you install an application, it's important to keep an eye on subsequent releases to make sure that you install updates that fix any security holes that arise. Deploying an application and not keeping it updated can leave you vulnerable not only to malicious users who want to break into your servers, but also spammers who look for problems with applications that can send mail to use them to deliver spam. The bottom line is that putting up an application on the Web places some responsibility on the person who deployed it to keep it up-to-date and prevent it from being used for nefarious purposes.

Q Is there a way to automatically keep my applications up-to-date?

A Some servers run operating systems with package managers. As long as you install your applications from packages supported by the package managers, you can let the operating system keep them up-to-date. Doing so generally requires that you run your own server. If you are using a web hosting provider, you're probably on your own when it comes to installing updates. Alternatively, you can use hosted applications. In that case, the company providing the applications is responsible for keeping them all up-to-date.

Q Will my web hosting provider install and maintain any of these applications for me?

A Some web hosting providers maintain installations of popular applications so that their customers don't have to install the applications themselves. For example, there's a list of web hosts that support WordPress at http://wordpress.org/hosting/. If you're sure which application you want to use, it might make sense to select your hosting provider on that basis.

Quiz

1. What are some of the trade-offs between hosted applications and those you install yourself?
2. Why do some applications require you to change the file permissions on the server?
3. Why do most installable applications ask you to specify a prefix for table names in a database?

Quiz Answers

1. Hosted applications tend to be less work up front and easier to maintain but offer less flexibility than applications you install.
2. Applications that make changes to the file system, either because they store their data in files or because they allow users to upload files, generally require you to change the file permissions for specific directories when you install them.
3. Each application has its own table prefix to prevent naming conflicts when several applications use the same database. For example, several applications might have their own table called "users." Adding an application-specific prefix to it will prevent conflicts.

Exercises

1. Find out which operating system, application development environment, and database are available from your web host, if you have one. If you're working on internal projects, find out about your servers from the Information Technology department.
2. Install one of the applications mentioned in this lesson, or one you find yourself, and try it out.
3. Go forth and put your new web publishing skills to good use.

This page intentionally left blank

APPENDIX A
Sources for Further Information

Haven't had enough yet? In this appendix, you'll find the URLs for all kinds of information about the World Wide Web, *Hypertext Markup Language (HTML)*, and developing websites. With this information, you should find just about anything you need on the Web.

CAUTION

Some of the URLs in this appendix refer to FTP sites. They might be busy during business hours, and you might not access the files immediately. Try again during nonprime hours.

Also, for mysterious reasons, some of these sites might be accessible through an FTP program but not through web browsers. If you're consistently refused by these sites using a browser, try an FTP program instead.

These sites are divided into the following categories and are listed in alphabetic order under each category:

- Analytics
- Browsers
- Collections of HTML and web development information
- Imagemaps
- HTML editors and converters
- HTML validators, link checkers, and simple spiders
- JavaScript
- Log file parsers
- HTML style guides
- Servers and server administration
- Sound and video
- Specifications for HTML, HTTP, and URLs
- Server-side scripting
- Web publishing tools
- Other web-related topics
- Tools and information for images
- Web hosting providers

Analytics

Google Analytics

http://www.google.com/analytics/

Yahoo! Web Analytics

http://web.analytics.yahoo.com/

Digits Counter

http://www.digits.com/

LiveCounter

http://www.chami.com/counter/classic/

Bravenet Counter

http://www.bravenet.com/webtools/counter/StatCounter

http://www.statcounter.com

Site Meter

http://www.sitemeter.com/

Browsers

Internet Explorer

http://www.microsoft.com/windows/ie/

Mozilla Firefox

http://www. mozilla.com

Safari

http://www.apple.com/safari/

Google Chrome

http://www.google.com/chrome

Opera

http://www.operasoftware.com

wget

http://www.gnu.org/software/wget/wget.html

IETester (allows testing multiple versions of IE)

`http://www.my-debugbar.com/wiki/IETester/HomePage`

Lynx (text-based browser)

`http://lynx.browser.org/`

Links (alternative to Lynx with better table handling)

`http://links.sourceforge.net/`

evolt Browser Archive (huge archive of old browsers)

`http://browsers.evolt.org/`

Collections of HTML and Web Development Information

The home of the WWW Consortium

`http://www.w3.org/`

MSDN (Microsoft Developer's Network) Online

`http://msdn.microsoft.com/`

Mozilla Developer Center

`http://developer.mozilla.org`

Yahoo! Developer Network

`http://developer.yahoo.com/`

Google Code

`http://code.google.com/`

evolt.org

`http://www.evolt.org/`

A List Apart

`http://www.alistapart.com/`

W3Schools

`http://www.w3schools.com/`

Smashing Magazine

`http://www.smashingmagazine.com/`

QuirksMode

`http://www.quirksmode.org/`

Imagemaps

Mapedit: A tool for Windows and X11 for creating imagemap map files

`http://www.boutell.com/mapedit/`

image-maps.com (online tool for creating image maps)

`http://www.image-maps.com/`

HTML Editors and Converters

HTML-Kit

`http://www.chami.com/html-kit/`

TopStyle

`http://www.topstyle4.com/`

Adobe Dreamweaver

`http://www.adobe.com/products/dreamweaver/`

BBEdit (Mac OS)

`http://www.barebones.com/`

HTML Validators, Link Checkers, and Simple Spiders

W3C Validator

`http://validator.w3.org/`

HTML Tidy

`http://tidy.sourceforge.net/`

WDG HTML Validator

`http://www.htmlhelp.com/tools/validator/`

CSE HTML Validator

`http://www.htmlvalidator.com/`

JavaScript

QuirksMode

http://www.quirksmode.org/

jQuery

http://jquery.com/

Dojo Toolkit

http://www.dojotoolkit.org/

YUI Library

http://developer.yahoo.com/yui/

YUI Compressor

http://developer.yahoo.com/yui/compressor/

Google Closure Tools

http://code.google.com/closure/

Douglas Crockford's World Wide Web

http://www.crockford.com/

script.aculo.us

http://script.aculo.us/

Log File Parsers

Analog

http://www.analog.cx/

Webalizer

http://www.mrunix.net/webalizer/

AWStats

http://awstats.sourceforge.net/

Mint

http://www.haveamint.com/

HTML Style Guides

Tim Berners-Lee's Style Guide

`http://www.w3.org/Provider/Style/Overview.html`

The Yale HyperText Style Guide

`http://www.webstyleguide.com/`

NYPL Online Style Guide

`http://legacy.www.nypl.org/styleguide/`

Servers and Server Administration

Apache (UNIX, Windows)

`http://httpd.apache.org/`

Tomcat (UNIX, Windows)

`http://jakarta.apache.org/tomcat/`

Current list of official MIME types

`http://www.iana.org/assignments/media-types/`

Microsoft Internet Information Server (Windows NT)

`http://www.microsoft.com/iis`

Sound and Video

Audacity

`http://audacity.sourceforge.net/`

Apple iTunes

`http://www.apple.com/itunes/`

WinAmp (free sound player for Windows)

`http://www.winamp.com/`

QuickTime

`http://www.apple.com/quicktime/`

RealNetworks

`http://www.realnetworks.com/`

Windows Media

`http://www.windowsmedia.com/`

VLC

`http://www.videolan.org/vlc/`

Microsoft Silverlight

`http://www.microsoft.com/silverlight/`

Specifications for HTML, HTTP, and URLs

HTML Specification and Info

`http://www.w3.org/MarkUp/`

HTML5

`http://html5.org/`

XHTML 1.0

`http://www.w3.org/TR/xhtml1/`

HTML 4.01

`http://www.w3.org/TR/html4/`

The HTTP specification (as defined in 1992)

`http://www.w3.org/Protocols/HTTP/HTTP2.html`

Cascading Style Sheets

`http://www.w3.org/Style/CSS/`

Information about HTTP

`http://www.w3.org/Protocols/`

Pointers to URL, URN, and URI information and specifications

`http://www.w3.org/Addressing/`

Server-Side Scripting

PHP

`http://www.php.net/`

ASP.NET

`http://www.asp.net/`

Web Publishing Tools

Blogger

`http://www.blogger.com/`

TypePad and Movable Type

`http://www.sixapart.com/`

WordPress

http://www.wordpress.com/

MediaWiki

`http://www.mediawiki.org/wiki/MediaWiki`

Drupal

`http://drupal.org/`

Tumblr

`http://www.tumblr.com`

Posterous

`http://posterous.com/`

Other Web-Related Topics

Adobe Acrobat

`http://www.adobe.com/products/acrobat/`

Web security overview

`http://www.w3.org/Security/Overview.html`

Tools and Information for Images

clipart.com

`http://www.clipart.com/`

Skitch

`http://skitch.com/`

IrfanView

`http://www.irfanview.com/`

GraphicConverter

http://www.lemkesoft.com/

Flickr Creative Commons

http://www.flickr.com/creativecommons/

Web Hosting Providers

DreamHost

http://www.dreamhost.com/

Pair.com

http://www.pair.com/

MediaTemple

http://mediatemple.com/

Index

Symbols

!= (inequality) operator, 422
 # (pound signs)
CSS and, 178
numbered entities, 150
% (modulus) operator, 418
% (percent sign)
URL escape codes, 122
in email addresses, 126
& (ampersand)
named entities, 150
troubleshooting, 152
*** (multiplication) operator, 418**
+ (addition) operator, 418
++ (increment) operator, 418
— (decrement) operator, 418
- (subtraction) operator, 418
.css file extension, 175
.htm versus .html, 58
/ (division) operator, 418
//, JavaScript comments, 414
== (equality) operator, 422

A

A List Apart.com, 694
<a> tag
accesskey attribute, 584
event handlers, 123
frameset attribute, 491-493, 496, 520
HTML 4.01 additions to, 123
name attributes and, 114
tabindex attribute, 123, 584
title attribute, 583
<abbr> tag, 133
absbottom alignment (images), 222
absmiddle alignment (images), 221
absolute pathnames, 106-107
linking local pages, 105-107
versus releative pathnames, 107
absolute positioning, page layout, 394-396
absolute positioning (CSS), 199-201
absolute units (CSS), 180-181
access counters, 693
access keys, forms, navigating, 342
accessibility, 579, 591
alternative browsers, 581
Building Accessible Websites (italic), 589
designing for, 586
color, 586
fonts, 586
forms, 587
frames, 587
HTML tags, 587

linked windows, 587
Dive Into Accessibility website, 589
myths, 579-580
Section 508, 580-581
validating, 588-589
W3C Web Accessibility Initiative, 589
writing HTML, 582
images, 585
links, 583-584
tables, 582-583
accesskey attribute (a tag), 584
accounts, Twitter, creating, 607
<acronym> tag, 134
action attribute (form tag), 313, 318-319
active state (links), 206
addition operator (+), 418
<address> tag, 147, 168, 561
addresses (Web). *See* URLs (uniform resource locators)
Adobe Acrobat, 699
Adobe Dreamweaver HTML editor, 695
Adobe Kuler, 236
Adobe Photoshop, 212
Adobe website, 212
advertising websites, 605
brochures, 612
business cards, 612
links from other sites, 606
social media, 606-609
AJAX, 433
external data, loading, 481-484
JQuery, 480-484
alert messages
displaying, 438
JavaScript, 431
align attribute (<caption> tag), 286
align attribute (<hr> tag), 144
align attribute (<iframe> tag), 522, 528
align attribute (<img> tag), 220-224, 255
align attribute (<object> tag), 373
align attribute (<table> tag), 282
alignment
see also *text alignment*
images, 220-222
tables, 282-283
captions, 286-287
cells, 283-286
columns, 296-299
rows, 300-302
alink attribute (<body> tag), 237-238
allowfullscreen attribute (<embed> tag), 375
allowscriptaccess attribute (<embed> tag), 375
alt attribute (<area> tag), 249
alt attribute (<img> tag), 215-216, 255, 584
alternative content, <object> tag, 374
alternative text to images, 215-216
Amazon.com, 597
ampersands (&)
named entities, 150
troubleshooting, 152
Analog website, 696
analytic websites, 693
anchor tag. See *<a> tags*
anchors, 113
creating, 113-114
hash signs and, 119
linking in same page, 120
sample exercises, 115-119
anonymous FTP, URLs, 124-125
anonymous functions, JavaScript, 447
Apache web servers, 697
Apple iTunes website, 697
Apple Safari, 9, 18, 693
Apple's QuickTime, container format, 362
archive attribute (<object> tag), 373
<area> tag, 248-249
alt attribute, 249
coords attribute, 249
href attribute, 249
shape attribute, 248
arguments
JavaScript, 416, 424
this, 432
arrays
JavaScript, 427
PHP, 624-626
associative arrays, 625
indexes, 624
array_key_exists() function, 626
array_values() function, 642
ASCII format, text editors, 57
Ask.com, 610
ASP.NET, 653
server-side scripting, 698
associating imagemaps with images, 249
associative arrays, PHP arrays, 625
asterisk (*), multiplication operator, 418
attributes
see also *CSS (cascading style sheets); specific attributes*
<a> tag
accesskey, 584
name, 114
tabindex, 123, 584
target, 491-493, 496, 520
title, 583

adding to Bookworm webpage sample, 166-167
<area> tag
alt, 249
coords, 249
href, 249
shape, 248
<audio> tag, 384
<body> tag
alink, 237-238
bgcolor, 236
link, 237-238
text, 236, 238
vlink, 237-238

 tag, clear, 226
case sensitivity, 56
<embed> tag, 375
<frame> tag, 527
bordercolor, 505-506
longdesc, 506
marginheight, 506
marginwidth, 506
<frameset> tag, 527
cols, 500-501, 514-515
rows, 501-502, 514-515
href attributes, links, 100
<iframe> tag, 528
align, 522
frameborder, 521
height, 521
hspace, 521
marginheight, 521
marginwidth, 521
name, 521
noresize, 521
scrolling, 521
src, 521
vspace, 521
width, 521
<img> tag, 255-256
align, 220-224
alt, 215-216, 584
border, 229
height, 232
hspace, 226-227
longdesc, 585
src, 215
usemap, 249-250
vspace, 226-227
width, 232
manipulating, jQuest, 473-474
<map> tag, name, 248
<object> tag, 373
style, 62
<table> tag, summary, 582
type attribute, 84, 87
value attributes, 87
<video> tag, 367
without values, 145
Audacity website, 697
audio, embedding, 383-385
audio players
Flash, embedding, 384-385
HTML tags for, 133-134
<audio> tag, 383-386
attributes, 384
authentication, 595
auto margins (CSS), 187
automatic updates, web applications, 688
autoplay attribute (audio tag), 384
autoplay attribute (video tag), 367
AWStats website, 696

B

<b> tag, 135, 168
background-attachment property (CSS), 239
background-color property, table background colors, changing, 280
background-position property (CSS), 239
backgrounds
background images, 238-240
colors, 236
CSS (cascading style sheets), 239
design tips, 555
elements, 241-242
img tag, 242
tables, background colors, changing, 280-282
bandwidth limitations, 598
base tag, 496-498, 527
baseline alignment (images), 222
BBEdit (Mac OS) HTML editor, 695
Better-Organized Style Sheet listing (13.7), 406-407
bgcolor attribute (<body> tag), 236
bgcolor attribute (<table> tag), 280
<big> tag, 135, 168
binding events, JQuery, 465
Bing, 610
blank target name, 520
block-level elements
backgrounds, 241-242
versus character-level elements, 132
<blockquote> tag, 147-148, 168
Blogger, 661
Blogger.com, 44, 699
blogging applications
TypePad, 667-669
WordPress, 669-673
blogging tools, 660-661
blogs, 28
blue_page window, 492-496
Bobby (accessibility validator), 588-589
body (tables), 301

<body> tag, 69-70
alink attribute, 237-238
bgcolor attribute, 236
CSS, applying, 205-206
CSS box properties, 188
link attribute, 237-238
text attribute, 236, 238
vlink attribute, 237-238
bold text, design tips, 541
bold type, headings as, 73
books, Building Accessible Websites, 589
Bookworm webpage code, 163-165
border attribute (<img> tag), 229, 256
border attribute (<object> tag), 373
border attribute (<table> tag), 270-273, 306
bordercolor attribute (<frame> tag), 505-506
bordercolordark attribute, 307
bordercolorlight attribute, 307
borders
around images that are links, 229
CSS box model, 183-184
frame borders, 504-506
image borders, 233
table borders, changing, 270-273
bottom alignment (images), 220
bottom property (CSS positioning), 196
box model (CSS), 182
borders, 183-184
clearing elements, 194
controlling size and element display, 189-192
floating elements, 192-195
margins, 185-189
padding, 185-189
**
 tag, 145-146, 169**
clear attribute, 226
table line breaks, 278-280
brackets, HTML tags, 56
Braille browsers, 581
Bravenet Counter, 693
breadcrumbnavigation, 572
brevity in webpage design, 539
brochures, including Web address on, 612
browsers. *See* Web browsers
***Building Accessible Websites*, 589**
built-in functions (PHP), 633
bullets, images, 243
business cards, including Web address on, 612
button attribute, <input> tag, 351
buttons. *See* form controls
<body> tags, CSS box properties and, 189
<button> tag, 351
form controls, 330-331

C

<caption> tag, 265-269, 305
align attribute, 286
captions, tables, 260
alignment, 286-287
creating, 265-269
carriage returns, 601
cascading (CSS), 179-180
Cascading Style Sheets (CSS). *See* CSS (Cascading Style Sheets)
case sensitivity
filenames, 601
HTML tags, 56
naming image files, 212
cellpadding attribute (<table> tag), 273-274, 307
cells (tables), 260
see also *columns (tables)*
alignment, 283-286
background colors, changing, 280-282
creating, 262-264
empty cells, creating, 264-265
padding, 273-274
spacing, 274-275
spanning multiple rows/columns, 287-289
cellspacing attribute, 307
cellspacing attribute (<table> tag), 274, 307
celsiusToFahrenheit() function, 649
** tag, 170**
char attribute (<table> tag), 303, 307
character entities, 149
encoding, 151
named entities, 150
numbered entities, 150
reserved characters, 152
character formatting, design tips, 541
character-level elements, 132
logical style tags, 132-134
physical style tags, 135-136
versus block-level elements, 132
charoff attribute (<table> tag), 303, 307
check box controls, creating with <input> tag, 325-326
checkbox attribute (<input> tag), 351
checked attribute (<input> tag), 325
radio, 327
checkform() function, 437
checking file permissions, 604
chmod command, 604
choosing Web servers, 595
bandwidth limitations, 598

domain parking, 597
IPPs (Internet presence providers), 596
ISPs (Internet service providers), 596
personal servers, 597
school servers, 595
Web presence providers, 596
work servers, 595

Chrome (Google), 9, 18

circles, imagemap coordinates, 247

<cite> tag, 168
as logical style tags, 133
versus <blockquote> tags, 147

clarity in web page design, 539

class attribute, 177-178

classes (jQuery), manipulating, 468-471

classes (CSS), 177-178
naming conventions, 208

classid attribute (<object> tag), 373

**clear attribute (
 tag), 226**

clear attribute (<img> tag), 255

clear property (CSS), 194

clearing text wraps, 225-226

client-side images, 243-244
area tag, 248-249
associating with images, 249
circle coordinates, 247
image selection, 244
imagemap creation software, 246
jukebox image example, 250-254
map tag, 248-249
polygon coordinates, 246-247
rectangle coordinates, 248
text-only browsers, 244
troubleshooting, 257

clients (FTP), 602

clipart.com, 699

closing HTML tags, 56, 70, 82, 262

cloud, content management systems, 658-659

cm unit (CSS), 181

code listings
13.1 (Using div Tags to Create Sections for Positioning), 392-393
13.2 (Style Sheet for Page Layout), 395
13.3 (Style Sheet for Colors and Fonts), 397-398
13.4 (Moving One Section Before Another), 399
13.5 (Float-Based Layouts in CSS), 401-402
13.6 (Randomly Organized Style Sheet), 405-406
13.7 (Better-Organized Style Sheet), 406-407

<code> tag, 168
as logical style tags, 133

codebase attribute (object tag), 373

codecs, 385
H.264 video codec, 362
converting to, 363-366

codetype attribute (object tag), 373

<col> tag, 297-298, 305

<colgroup> tag, 296-298, 305

color pickers, 234

color property, table background colors, changing, 280

color property (CSS), 236

Color Schemer, 235

colors, 234
accessibility, 586
background colors, 236
color pickers, 234
CSS properties, 238
naming, 234-235
page layout, 397-398
tables, background colors, 280-282
text colors, 236-237

cols attribute (<frameset> tag), 500-501, 514-515, 527

colspan attribute (<td> tag), 288, 308

colspan attribute (<th> tag), 288, 308

columns (tables)
see also *cells (tables)*
aligning, 296-299
grouping, 296-299
width of, 275-277
percentages as, 270

comments, 75-76
HTML tags, 79
JavaScript, 414
PHP, 622-623

community publishing applications, 661

company profiles, websites, 29

comparison operators (JavaScript), 421-422

compatibility, browsers, 387

complex framesets, creating, 507-520

conditional operators (PHP), 629-630

conditional statements (PHP), 628-629

confirmation file, WordPress, 670

consistent page layout, 545

container formats, video, 362-363

content (websites), 32
adding, jQuest, 474-477
adding to sample pages, 158-159
alternative content, <object> tag, 374
Facebook, embedding, 685-686

formatting. *See* HTML tags
goal setting, 30-31
ideas for, 28-30
main topics, determing, 31-32
modifying, jQuery, 468-477
navigation, 32-42
organization, 32-42
hierarchical, 33-35
hierarchical/linear, 38-39
linear, 35-36
linear with alternatives, 36-37
web, 39-42
removing, jQuest, 474-477
splitting topics across pages, 558
storyboarding, 42-44
structures, 33
titles, 71
Twitter, embedding, 684-685
web pages, adding to, 452-455

content management systems, 44-45, 657-660
cloud, 658-659
data repositories, 663
Drupal, 677-682
templating systems, 663
types of, 660
blogging tools, 660-661
community publishing applications, 661
general-purpose content management systems, 663
image galleries, 662
wikis, 661-662
workflow systems, 663

content pages, creating for framesets, 507-514

contextual selectors (CSS), 176-177

control structures, JavaScript, 421-424

controlling loop execution, PHP, 632-633

controls (forms), 312

controls attribute (audio tag), 384

controls attribute (video tag), 367

converter programs for HTML, 63-64

converters, websites, 695

cookies (PHP), 652

coordinates (imagemaps)
area tag, 248-249
circle coordinates, 247
determining, 245-246
imagemap creation software, 246
map tag, 248-249
obtaining from browsers, 251
polygon coordinates, 246-247
rectangle coordinates, 248

coords attribute (<area> tag), 249

cross-platform Web compatibility, 10
reasons for, 52

cross-site scripting, preventing, 636-637

CRUD applications, 642

CSE HTML Validator, 695

CSS (Cascading Style Sheets), 22, 45, 389, 411
advantages, 61
applying, 174
background properties, block-level elements, 241-242
backgrounds, <img> tag, 242
box model, 182
borders, 183-184
clearing elements, 194
controlling size and element display, 189-192
floating elements, 192-195
margins, 185-189
padding, 185-189
cascading, 179-180
creating, 61-62
CSS files, creating, 175
external files, placing in, 556-558
font properties, 138-139, 155-156
form properties, applying styles, 345-349
forms, applying to, 343-345
HTML tags, combining with, 62
layout style sheets, writing, 394-401
links, modifying, 206-207
list-style-image properties, unordered lists, 89
list-style-position properties, unordered lists, 89
overview, 173
page-level styles, creating, 174
positioning, 196-197
absolute positioning, 199-201
relative positioning, 197-198
static positioning, 196
top/left/bottom/right properties, 196
z-index property (stacking), 202-205
properties
background-attachment, 239
background-position, 239
color, 236
font-family, 170
font-size, 170
font-style, 170
font-variant, 170

font-weight, 170
retrieving, 467-468
text-align, 170
text-decoration, 170
sample pages, creating, 157
adding attributes, 166-167
adding content, 158-159
adding tables of contents, 159
Bookworm web page code, 163-165
frameworks, 157-158
link menus, 161-162
page descriptions, 160
planning the page, 157
signatures, 163
testing results, 165-166
unordered lists, 162-163
selectors, 176
classes, 177
contextual selectors, 176-177
IDs, 177-178
sitewide-style sheets, creating, 175-176
spacing units, 156
specifications website, 698
standards, 208
tables, laying out, 390-402
text-decoration properties, 137
troubleshooting, 207
units of measure, 180-181
URLs (uniform resource locators), using, 182
web design, 403-404
organization, 404-406
site-wide style sheets, 407
web designs, 403-404
customizing videos, YouTube, 359-360
CuteFTP, 602
Cynthia Says validator, 588-589

D

Dashboard (Google Analytics), 615-616
data attribute (object tag), 373
data repositories, content management, 663
data types, JavaScript, 426
database connectivity (PHP), 651
databases, relational databases, 664-666
<dd> tag, 82, 91, 96
declare attribute (object tag), 373
decrement operator (—), 418
default index files, 599-600
definition terms. See <dt> tags
deploying
packaged software, 666
PHP files, 654
descriptive titles, 541
design
designing websites, 538
accessibility, 579-580, 586-587
backgrounds, 555
brevity, 539
browser-specific terminology, 542
clarity, 539
consistent layout, 545
determining user preferences, 573-575
emphasis, 541, 587
first time users versus regular users, 573
frames, 570
goal setting, 30-31
grouping related information, 544
hardware considerations, 555
headings, 543-544
hierarchical, 33-35
hierarchical/linear, 38-39
images, 552
image size, 553-554
when to use, 552-553
linear, 35-36
linear with alternatives, 36-37
links, 546-558
main topics, determining, 31-32
navigation, 572-573
number of pages, 558-561
organizing for quick scanning, 539-540
page signatures, 561-562
page validation, 535-538
proofreading, 543
search engines, 569-570
spell checking, 543
splitting topics across pages, 558
standalone pages, 541
standards compliance, 532-538
storyboarding, 42-44
style sheets, 403-404
organization, 404-406
site-wide, 407
URLs, 570-572
user experience levels, 569
user preferences, 568-569
web, 39-42
XHTML, 579
development environments, JavaScript, 433
dfn tag, 168
as logical style tags, 133
DHTML (Dynamic HTML), 543
dictionaries (PHP), 625

Digits Counter, 693
<dir> tags, 82
directories, 599
folders and, 105
index files, web servers, 599
URLs, 122
WordPress, 670
disabled form controls, creating, 342-343
discographies, 32
displaying alert messages, 438
Disqus, 659
distributed nature of the Web, 10-11
** tag, 153-155, 170, 393**
Dive Into Accessibility website, 589
division operator (/), 418
<dl> tag, 91,96
Doctorow, Cory, 11
DOCTYPE identifiers, 68
<DOCTYPE> tag, 58
DOCTYPEs (HTML5), 577
document object (JavaScript), 429
document roots, 107
document type definition (DTD), Internet Explorer, 188
documentation, JavaScript, 432
documents, web pages, converting to, 564
documents (HTML)
colors, 234
background colors, 236
CSS properties, 238
naming, 234-235
text colors, 236-237
frameset documents, 498-499
images
adding, 214-218
aligning with text, 220-222
alternative text, 215-216
as links, 228-231
background images, 238-240
borders, 233
bullets, 243
GIF (Graphics Interchange Format), 213, 256
Halloween House web page example, 217-218
height/width, 232
image etiquette, 254-255
inline images, 214-215, 219
JPEG (Joint Photographic Experts Group), 213, 256
navigation icons, 229-231
PNG (Portable Network Graphics), 214
preparing for Web, 212
scaling, 232
spacing around, 226-228
wrapping text around, 223-226
linking to frames, 502-503, 518
links
colors, 555
definitions as links, 551
explicit navigation links, 550
"here" links, 549-550
home page links, 558
implicit navigation links, 550
link menus, 546-547
text links, 547-548
when to use, 550, 552
moving between Web servers, 600-603
organizing, 598
organizing
default index files, 599-600
directories, 599
filenames, 599-600
Web server setup, 598-599
style guides, 697
text
aligning images with, 220-222
colors, 236-237
wrapping around images, 223-224, 226
text formatting, 541
XHTML 1.0 specification, 534
Dojo JavaScript library, 461
Dojo Toolkit, 696
DOM
accessing
JavaScript methods, 454
methods, 448
navigating, node properties, 450
domain names, registering, 596
domain parking, 597
domains, 596-597
dotster.com, 597
Douglas Crockford's Wrrrld Wide Web, 696
downloading MediaWiki, 675-676
DreamHost web hosting, 700
Drupal, 661, 677-682, 699
Drupal Gardens.com, 45
<dt> tag, 82, 91, 96
DTD (document type definition), Internet Explorer, 188
Dunbar Project example
absolute positioning, 394-396
colors/fonts, 397-398
floated columns, 401-402

HTML sections, 391-394
re-ordering sections, 398-399
redesigning layout, 400-401
style sheet, 394-398
tables, 390-391
Dynamic HTML (DHTML), 543
dynamic nature of Web, 11-12

E

echo() function, 643
editing PHP files, 654
editors (HTML), 695
educational websites, 29
element displays, CSS box model, controlling, 189-192
elements (HTML), 56
backgrounds, 241-242
hiding, JQuery, 466-467
JavaScript, hiding/showing, 443-451
showing, JQuery, 466-467
***Elements of Style, The*, 539**
em unit (CSS), 180
** tag, 116, 168**
as logical style tags, 132
em values, 409
email, sending, PHP, 652
<embed> tag, 375-376, 386
attributes, 375
embedding
audio, 383-385
content
Facebook, 685-686
Twitter, 684-685
Flash multimedia
object tag, 370-374
SWFObject, 376-377
sound/video, 375
video, 356-361, 366-370
empty cells, creating, 264-265
encoding character entities, 151
enctype attribute (<form> tag), 320, 350
file, 330
Enterprise Edition (Java), 653
equality operator (==), 422
escape codes (URLs), 122
HTML reserved characters, 152
event handlers
<a> tags, 123
JavaScript, 430
onblur, 430
onchange, 430
onclick, 430
onfocus, 430
onload, 430
onmouseover, 430
onselect, 430
onsubmit, 430, 437
onunload, 430
event-driven models of execution, JavaScript, 412
events
binding, JQuery, 465
event handlers, onsubmit, 437
JavaScript, 429-432
evolt Browser Archive, 694
evolt.org, 694
ex unit (CSS), 180
exercises
Lesson 1, 23
Lesson 2, 48
Lesson 3, 65-66
Lesson 4, 80
Lesson 5, 98
Lesson 6, 130
Lesson 7, 172
Lesson 8, 209
Lesson 9, 258
Lesson 10, 310
Lesson 11, 353
Lesson 12, 388
Lesson 13, 410
Lesson 14, 434
Lesson 15, 457
Lesson 16, 487
Lesson 17, 530
Lesson 18, 566
Lesson 19, 592
Lesson 20, 618
Lesson 21, 655
Lesson 22, 689
expand all/collapse all links, FAQs, adding to, 452
expressions, JavaScript, 417-418
eXtensible HyperText Markup Language (XHTML). *See* XHTML (eXtensible HyperText Markup Language)
eXtensible Markup Language (XML). *See* XML (eXtensible Markup Language)
extensions, image files, case sensitivity, 212
external data, AJAX, loading, 481-484
external files
CSS, placing in, 556-558
JavaScript, placing in, 556-558
websites, 591

F

Facebook
advertising websites via, 606-609
content, embedding, 685-686
fan sites, 29
FAQs (frequently asked questions), expand all/collapse all links, adding to, 452
Fetch, 602

<fieldset> tag, 351
file attribute (<input> tag), 351
file extensions, 600
file formats
GIF (Graphics Interchange Format), 213, 256
JPEG (Joint Photographic Experts Group), 213, 256
PNG (Portable Network Graphics), 214
File Not Found errors, 603
File Transfer Protocol (FTP). *See* FTP (File Transport Protocol)
file upload controls, creating with <input> tag, 329-330
file uploads (PHP), 652
file URLs, 126-127
filenames, 599-601
files
determining type of, 594
downloading, FTP sites, 19
file extensions, 600
File Not Found errors, 603
filenames, 601
Forbidden errors, 603
GIF (Graphics Interchange Format), 213, 256
HTML files, 55-60
index.html files, creating, 599-600
JPEG (Joint Photographic Experts Group), 213, 256
log files, 613
Google Analytics, 614
managing, 594
moving between Web servers, 600
carriage returns/line feeds, 601
filename restrictions, 601
FTP (File Transfer Protocol), 601-603
organizing, 598
default index files, 599-600
directories, 599
filenames, 599-600
Web server setup, 598-599
permissions, checking, 604
PHP files, deploying and editing, 654
PNG (Portable Network Graphics), 214
server-side processing, 595
Firefox. *See* Mozilla Firefox
fixed layouts, 395
Flash
audio players, embedding, 384-385
interactivity of, 14
multimedia, embedding, 370-377
video players, 378-381
flashvars (embed tag), 375
Flickr, 682-684
Flickr Creative Commons, 700
Float-Based Layouts in CSS listing (13.5), 401-402
floated columns layout, 401-402
floating elements, CSS box model, 192-195
floating frames, 54, 522
Flowplayer, 380-381, 384
folders, 105
font properties (CSS), 138-139, 155-156
font-family property (CSS), 170
font-size property (CSS), 170
font-style property (CSS), 170
font-variant property (CSS), 170
font-weight property (CSS), 170
fonts
accessibility, 586
page layout, 397-398
sans-serif fonts, 62
serifs, 62
footers (tables), 301
for loops, 423
JavaScript, 422-423
PHP, 631
Forbidden errors, 603
foreach loops (PHP), 630-631
form attribute (object tag), 373
form controls
<button> tag, 330-331
creating with <input> tag, 321-322
check box controls, 325-326
file upload controls, 329-330
generic buttons, 328
hidden form fields, 329
password controls, 323-324
radio buttons, 326-327
reset buttons, 325
submit buttons, 324
text controls, 322-323
disabled controls, creating, 342-343
<fieldset> tag, grouping controls, 340-341
<form> tags, 317
displaying label elements, 320-321
<legend> tag, grouping controls, 340-341
naming, 315
navigation
access keys, 342
default navigation, changing, 341-342
<option> tag, menus, creating, 332-334
readonly controls, creating, 342-343

registration form example, 334, 337-339
submit buttons, images as, 327-328
<select> tag, menus, creating, 332-334
<textarea> tag, 331-332
<form> tag, 312, 350
action attribute, 313, 350
enctype attribute, 320, 350
get method, 313-314
method attribute, 313-314, 319-320, 350
post method, 313-314
formatting text, HTML pages, 60, 64
formatting content. *See* HTML tags
forms (HTML), 13-14, 311-313
accessibility, 587
check box controls, creating with <input> tag, 325-326
controls, 312
creating, 312-317
CSS (cascading style sheets), applying to, 343-345
CSS properties, applying styles, 345-349
file upload controls, creating with <input> tag, 329-330
form controls
creating with <input> tag, 321-322
naming, 315
<form> tag
action attribute, 313
get method, 313-314
method attribute, 313-314
post method, 313-314
<form> tag
form controls, 317
using, 317-320
generic buttons, creating with <input> tag, 328
hidden form fields, creating with <input> tag, 329
input tag, 314-315
interactivity of, 13-14
navigation
access keys, 342
default navigation, changing, 341-342
password controls, creating with <input> tag, 323-324
PHP, 636-637
parameters with multiple values, 637-638
presenting forms, 642-645, 647
validating forms, 638-642
planning, 349-350
radio buttons, creating with <input> tag, 326-327
registration form, 14
required fields, testing, 438-439
reset buttons, creating with <input> tag, 325
security, 343
submit buttons
creating with <input> tag, 324
images as, 327-328
tags, 312
testing, 352
text controls, creating with <input> tag, 322-323
text form controls, 314
validating with JavaScript, 436
checkform() function, 437
code listing, 439-443
onsubmit event handler, 437
selected variable, 438
thisform object, 438
values, manipulating, 471-473
forms processing, server-side scripts, 594
frame attribute, 307
frame tag, 502-503, 527-529
bordercolor attribute, 505-506
longdesc attribute, 506
marginheight attribute, 506
marginwidth attribute, 506
name attribute, 506
noresize attribute, 506
scrolling attribute, 506-507
src attribute, 507
frameborder attribute (<frameset> tag), 528
frameborder attribute (<iframe> tag), 521, 528
frames, 489-491, 498
accessibility, 587
adding to web pages, 570
borders, 504-506
browser support for, 490
detriments, 527
<frame> tag, 502-503, 527
bordercolor attribute, 505-506
longdesc attribute, 506
marginheight attribute, 506
marginwidth attribute, 506
name attribute, 506
noresize attribute, 506
scrolling attribute, 506-507
src attribute, 507
framesets, 491
base tag, 496-498
combining vertical and horizontal frames, 514-515
content pages, 507-514
creating, 491-500

creating complex, 507-520
frameset documents, 498-499
frameset tag, 499-502
horizontal frames, 501-502
nesting, 516
vertical frames, 500-501
horizontal frames, 501-502
combining with vertical frames, 514-515
inline frames, 521-523
linked windows, 491-495
linking documents to, 502-503, 518
magic target names, 520
margins, 506
names, 506
naming, 517
<noframes> tag, 503-504, 519-520, 528
scrolling, 506-507
vertical frames, 500-501
combining with horizontal frames, 514-515

frameset documents, 498-499

Frameset specification (XHTML), 534

<frameset> tag, 499-500, 527, 529
cols attribute, 500-501, 514-515
nesting, 516
rows attribute, 501-502, 514-515

framesets, 54, 491
<base> tag, 496-498
combining vertical and horizontal frames, 514-515
content pages, 507-514
creating, 491-496, 499-500
creating complex, 507-520
frameset documents, 498-499
<frameset> tag, 499-502
horizontal frames, 501-502
nesting, 516
vertical frames, 500-501

frameworks, HTML files, creating, 157-158

FTP (File Transfer Protocol), 19, 46, 601-603
anonymous FTP, URLs, 124-125
navigating FTP servers, 125
non-anonymous FTP, URLs, 125

FTP Explorer, 602

functions
array_key_exists(), 626
celsiusToFahrenheit(), 649
checkform(), 437
echo(), 643
is_array(), 626
JavaScript, 424-426
anonymous functions, 447
PHP
built-in functions, 633
user-defined functions, 634-635
popup(), 524-525
related_posts(), 674
strip_tags(), 644
unset(), 625

<fieldset> tag, grouping controls, 340-341

<form> tag
action attribute, 318-319
enctype attribute, 330
form controls and, 317
using, 3-320

G

general-purpose content management systems, 663

generic buttons, creating with <input> tag, 328

get method (<form> tag), 313-314, 319-320

getElementsByClassName() method, 460

GIF (Graphics Interchange Format), 213, 256

glossary lists, 90

goal setting (website design), 30-31

godaddy.com, 597

Google, 609-610

Google Analytics, 614, 693
Dashboard, 615-616
installing, 614
reports, 615-616

Google Chrome, 9, 18, 693

Google Closure Tools, 696

Google Code, 694

Google Web Toolkit JavaScript library, 462

graphical navigation on the Web, 9

GraphicConverter, 700

Graphics Interchange Format (GIF), 213, 256

greater than operator, 422

greater than or equal to operator, 422

grouping table columns, 296-299

grouping related information, 544

H

H.264 codec, 362
converting to, 363, 365-366
versus Ogg Theora, 387

Halloween House web page, 216-218
hash signs, anchors and, 119
hashes (PHP), 625
<head> tag, 69
heading levels (HTML tags), 72-74
style attributes and, 62
headings (tables), 300-301
boldface, 73
design tips, 543-544
importance of, 539
tables, 260
height, images, 232
height attribute (<embed> tag), 307, 375
height attribute (<iframe> tag), 307, 521, 528
height attribute (<img> tag), 232, 554
height attribute (<object> tag), 373
height attribute (<video> tag), 367
here links, avoiding, 549-550
hidden attribute, <input> tag, 351
hidden form fields, creating with <input> tag, 329
hiding elements, JavaScript, 443-451
hierarchical organization, websites, 33-35
hierarchical/linear organization, websites, 38-39
history object (JavaScript), 429
hits, websites, 605
hobby-themed sites, 29
home pages, 26-27, 48
browsers, 26
ideas for, 27
initial visit, 27
linking to, 558
horizontal frames, 501-502
combining with vertical frames, 514-515
horizontal rules. *See* <hr> tags
hosting videos, 361-366
YouTube, 357
hostnames (URLs), 121-122
hover state (links), 206
<hr> tag, 142-144, 169
href attributes
area tag, 249
links, creating, 100
hspace attribute (iframe tag), 521
hspace attribute (img tag), 226-227, 256
hspace attribute (object tag), 373
<html> tag, 68-69
HTML (Hypertext Markup Language), 15, 22, 26, 45, 50, 691
adaptibility, reasons for, 52
as structurer, 50-51
converter programs, 63-64
DHTML (Dynamic HTML), 543
files, 55-60
formatting text, 64
HTML5, 55
ISO-Latin-1 character set, 150
links. See links, 100
overview, 50
page layout sections, 391-394
page layouts and, 51
parsing, 51
programs for writing, 62, 64
sample pages, creating, 59
specifications websites, 698
structure, writing with, 391-393
structuring, 68
Unicode, 151
versus XHTML, 534-535
writing accessible HTML, 582
images, 585
links, 583-584
tables, 582-583
HTML 4.01, 579
website, 698
HTML documents
colors, 234
background colors, 236
CSS properties, 238
naming, 234-235
text colors, 236-237
frameset documents, 498-499
HTML 4.01 specification, 534
images
adding, 214-218
aligning with text, 220-222
alternative text, 215-216
as links, 228-231
background images, 238-240
borders, 233
bullets, 243
GIF (Graphics Interchange Format), 213, 256
Halloween House web page example, 217-218
height/width, 232
image etiquette, 254-255
inline images, 214-215, 219
JPEG (Joint Photographic Experts Group), 213, 256
navigation icons, 229-231
PNG (Portable Network Graphics), 214
preparing for Web, 212

scaling, 232
spacing around, 226-228
wrapping text around, 223-224, 226
links
colors, 555
definitions as links, 551
explicit navigation links, 550
"here" links, 549-550
home page links, 558
implicit navigation links, 550
link menus, 546-547
text links, 547-548
to frames, 502-503, 518
when to use, 550, 552
moving between Web servers, 600
carriage returns/line feeds, 601
filename restrictions, 601
FTP (File Transfer Protocol), 601, 603
organizing, 598
default index files, 599-600
directories, 599
filenames, 599-600
Web server setup, 598-599
text
aligning images with, 220-222
colors, 236-237
wrapping around images, 223-224, 226
text formatting, design tips, 541

HTML editors, 695

HTML elements. *See* HTML tags

HTML forms. *See* forms

HTML pages
creating, 57-59
viewing results, 59-60
links, creating, 101-103
local pages, linking, 105-107
text formatting, 60

HTML style guides, 697

HTML tags, 51, 68, 78, 95
<a>
accesskey attribute, 584
event handlers, 123
HTML 4.01 additions to, 123
links, creating, 100-101
name attributes and, 114
tabindex attribute, 123, 584
target attribute, 491-493, 496, 520
title attribute, 583
<abbr>, 133
accessibility, 587
<acronym>, 134
<address>, 147, 168, 561
<area>, 248-249
alt attribute, 249
coords attribute, 249
href attribute, 249
shape attribute, 248
<audio> tag, 383-384, 386
<b>, 135, 168
<base>, 496-498, 527
<big>, 135, 168
<blockquote>, 147-148, 168
<body>, 69-70
alink attribute, 237-238
bgcolor attribute, 236
link attribute, 237-238
text attribute, 236, 238
vlink attribute, 237-238

, 169, 226
clear attribute, 226
table line breaks, 278
brackets, 56
<button>, 351
form controls, 330-331
<caption>, 265-269, 305
align attribute, 286
case sensitivity, 56
, 170
character entities, 149
encoding, 151
named entities, 150
numbered entities, 150
reserved characters, 152
character-level elements, 132
character-level elements versus block-level elements, 132
<cite>, 168
as logical style tags, 133
versus <blockquote>, 147
closing, 56, 70, 82
<code>, 168
as logical style tags, 133
<col>, 297-298, 305
<colgroup>, 296-298, 305
comments, 75-76, 79
conventions, founding of, 52
CSS (cascading style sheets), combining with, 62
<dd>, 82, 91, 96
designing web pages, 254-255
<dfn>, 168
as logical style tags, 133
<dir>, 82
, 153-155, 170
<dl>, 91, 96
DOCTYPE identifiers, 68
<dt>, 82, 91, 96
, 116, 168
as logical style tags, 132
<embed>, 375-376, 386
<fieldset>, 351
<form>, 350
<frame>, 502-503, 527-529

bordercolor attribute, 505-506
longdesc attribute, 506
marginheight attribute, 506
marginwidth attribute, 506
name attribute, 506
noresize attribute, 506
scrolling attribute, 506-507
src attribute, 507
<frameset>, 499-500, 527-529
cols attribute, 500-501, 514-515
nesting, 516
rows attribute, 501-502, 514-515
<head>, 69
heading levels, 72, 74
style attributes and, 62
<hr>, 142, 143, 169
attributes, 143-145
<html>, 68-69
<i>, 116, 135, 168
<iframe>, 521-523, 528
<img>, 214-215, 255-256
align attribute, 220-224
alt attribute, 215-216, 584
border attribute, 229
height attribute, 232
hspace attribute, 226-227
longdesc attribute, 585
src attribute, 215
usemap attribute, 249-250
vspace attribute, 226-227
width attribute, 232
indented code, 73
<input>, 350-351
<kbd>, 168
as logical style tags, 133
<label>, 314, 351
displaying label elements, 320-321
<legend>, 351
, 82, 86-87, 95
<link>, 175
<lists>, 82-83
glossary lists, 90
nesting lists, 92-93
numbered lists, 83
numbered lists, customizing, 85-87
unordered lists, 88
unordered lists, customizing, 88-89
logical style tags, 132-134
<map>, 248-249
name attribute, 248
markup languages, 53
meta tags, 611
nested, 70, 105
<nobr>, 169
<noembed>, 375
<noframes>, 503-504, 519-520, 528
<object>, 370-374, 382, 386
, 82-83, 95
type attributes and, 86
opening, 56
<option>, 351
origins, 53-54
<p>, 169, 385
<paragraph>, 75
<param>, 386
physical style tags, 135-136
<pre>, 139-141, 168
<s>, 135, 168
<samp>, 133, 168
sample pages, creating, 76-77, 157
adding attributes, 166-167
adding content, 158-159
adding tables of contents, 159
Bookworm web page code, 163-165
frameworks, 157-158
link menus, 161-162
page descriptions, 160
planning the page, 157
signatures, 163
testing results, 165-166
unordered lists, 162-163
<script>, JavaScript, 414-415
<select>, 351
<small>, 135, 168
<source>, 386
, 137, 168
, 132, 168
<style>, type attribute, 174
<sub>, 135, 168
<sup>, 135, 168
<table>261, 305
align attribute, 282
border attribute, 270-273
cellpadding attribute, 274
cellspacing attribute, 274
summary attribute, 261-262, 582
width attribute, 269-270
<tbody>, 301, 305
colspan attribute, 288
nowrap attribute, 279
rowspan attribute, 288
valign attribute, 284
<td>, 262, 305
<textarea>, 331-332, 351
text alignment, 153
blocks of elements, 153, 155
individual elements, 153
<tfoot>, 301, 305
<th>, 262, 288, 305, 582
<thead>, 300
<thread>, 305

<title>, 70-71
<tr>, 262, 305
<tt>, 135, 168
<u>, 135, 168
, 82, 95
<var>, 168
as logical style tags, 133
<video>, 366-370, 382, 386
<wbr>, 169
XHTML 1.0 and, 55
HTML Tidy, 538, 695
HTML validators, 695
Cynthia Says, 588-589
HTML-Kit text editor, 57, 695
HTML5, 55, 535, 577-578, 590
benefits, 575-576
deficiencies, 577
DOCTYPEs, 577
migrating to, 575
XML, 578
HTML5.org, 698
HTTP (Hypertext Transfer Protocol), 19, 121
specifications website, 698
URLs, 123-124
HTTP servers. *See* Web servers
hyperlinks, images, 228-231
hypertext. *See* links
Hypertext Markup Language (HTML). *See* HTML (Hypertext Markup Language)
hypertext references. *See* href attributes
Hypertext Transfer Protocol (HTTP). See HTTP (Hypertext Transfer Protocol)

I

<i> tag, 116, 135, 168
icons (navigation), 229-231
id attribute, 177-178
IDs (CSS), 177-178
naming conventions, 208
IE (Internet Explorer). *See* Internet Explorer
IETester, 694
if statement (JavaScript), 421-422
<iframe> tag, 521-523, 528
image attribute, <input> tag, 351
image galleries, 662
image-maps.com, 695
imagemaps, 695
client-side imagemaps, 243-244
area tag, 248-249
associating with images, 249
circle coordinates, 247
image selection, 244
imagemap creation software, 246
jukebox image example, 250-254
map tag, 248-249
polygon coordinates, 246-247
rectangle coordinates, 248
text-only browsers, 244
troubleshooting, 257
coordinates, determining, 245-246
server-side imagemaps, 243-244
images, 212
adding to web pages, 216-218
aligning with text, 220-222
alternative text, 215-216
as links, 228-231
background images, 238-240
borders, 229, 233
bullets, 243
client-side imagemaps, 243
area tag, 248-249
associating with images, 249
circle coordinates, 247
image selection, 244
imagemap creation software, 246
jukebox image example, 250-254
map tag, 248-249
polygon coordinates, 246-247
rectangle coordinates, 248
text-only browsers, 244
troubleshooting, 257
coordinates, getting from browsers, 251
extensions, case sensitivity, 212
GIF (Graphics Interchange Format), 213, 256
Halloween House web page example, 216-218
height/width, 232
image etiquette, 254-255
image size, 553-554
<img> tag, 214-215
inline images, 214-215, 219
JPEG (Joint Photographic Experts Group), 213-214, 256
navigation icons, 229-231
PNG (Portable Network Graphics), 214
scaling, 232
server-side imagemaps, 243-244
spacing around, 226, 228
submit buttons, creating with, 327-328
tiling, 238
tools, 699
troubleshooting, 604
when to use, 552-553

wrapping text around, 223-224, 226
writing accessible HTML, 585
<img> tag, 214-215, 255-256
align attribute, 220-224
alt attribute, 215-216, 584
border attribute, 229
CSS backgrounds, 242
height attribute, 232, 554
hspace attribute, 226-227
longdesc attribute, 585
src attribute, 215
usemap attribute, 249-250
vspace attribute, 226-227
width attribute, 232
important class, 177-178
in unit (CSS), 181
includes (PHP), 647-650
increment operator (++), 418
indented HTML code, 73
index files, 599-600
indexes
PHP arrays, 624

inequality operator (, =), 422
inline frames, 521-523
<input> tag, 314-315, 325
button attribute, 351
checked attribute, 325, 351
radio, 327
file attribute, 351
form control creation, 321-322
check box controls, 325-326
file upload controls, 329-330
generic buttons, 328
hidden form fields, 329
password controls, 323-324
radio buttons, 326-327
reset buttons, 325
submit buttons, 324
submit buttons, images as, 327-328
text controls, 322-323
checkbox attribute, 351
hidden attribute, 351
image attribute, 351
password attribute, 350
radio attribute, 351
reset attribute, 351
submit attribute, 350
text attribute, 350
type attribute, 350
button, 328
file, 329
hidden, 329
radio, 326
value attribute, 323, 325
installation page, WordPress, 671
installing
Google Analytics, 614
MediaWiki, 675-676
web applications, 688
Internet media types, 372
Internet Explorer, 9, 15, 693
document type definition (DTD), 188
Internet presence providers (IPPs), 596
Internet service providers (ISPs), 596, 617
intranets, 9, 26
IPPs (Internet presence providers), 596
IrfanView, 212
IrfanView.com, 699
ISO-Latin-1 character set, 150
ISPs (Internet service providers), 596, 617
is_array() function, 626
italic text, design tips, 541
iterations, loops, 422
iTunes website, 697

J

Java Enterprise Edition, 653
JavaScript, 12, 22, 412
advantages of, 412-413, 432
alert message, 431
anatomy of scripts, 414
arguments, 416, 424
arrays, 427
browsers
compatibility, 433
integration, 413
comparison operators, 421
control structures, 421-424
data types, 426
development environments, 433
document object, 429
documentation, 432
DOM, accessing, 448
ease of use, 412-413
elements, hiding/showing, 443-451
environment, 428-429
event handlers, 430
event-driven models of execution, 412
events, 429-432
expressions, 417-418
external files, placing in, 556-558
for loops, 423
form validation, 436
checkform() function, 437
code listing, 439-440, 442-443
onsubmit event handler, 437
selected variable, 438

thisform object, 438
forms, required fields, 438-439
functions, 424-426
anonymous functions, 447
history object, 429
if statement, 421-422
interactivity of, 14
libraries, 459-460
Dojo, 461
enabling users, 485
Google Web Toolkit, 462
jQuery, 461-484
Midori, 462
MochiKit, 462
MooTools, 462
Prototype, 461-462
slowly loading pages, 485
Yahoo, UI (YUI), 461
linked windows, opening, 523-526
location object, 429
looping statements, 422-423
objects, 427-428
operators, 417-418
parameters, 424
reserved words, 419
<script> tag, 414-415
server efficiency, 413
syntax, 415-428, 433
validation function, 437
variables, 418-420
web pages
adding content to, 452-455
integrating, 456
websites, 696
while loops, 423-424
window object, 429

Joint Photographic Experts Group (JPEG), 213, 256

journals (web), 28

JPEG (Joint Photographic Experts Group), 213, 256

jQuery JavaScript library, 461-463, 696
adding/removing content, 474-477
AJAX, 480-484
Attributes, manipulating, 473-474
binding events, 465
classes, manipulating, 468-471
enabling users, 485
form values, manipulating, 471-473
hiding and showing elements, 466-467
modifying page styles, 466-468
modifying web page content, 468-477
retrieving style sheet properties, 467-468
sample script, 463-464
slowly loading pages, 485
special effects, 478-480

jukebox imagemap, 250-254

JW Player, 378-380, 384

K

<kbd> tag, 168
as logical style tags, 133

Konqueror, 18

<label> tag, 314, 351
displaying label elements, 320-321

L

layout, 543
absolute positioning, 394-396
colors/fonts, 397-398
consistent layout, 545
CSS, positioning, 196-205
fixed, 395
floated columns, 401-402
group-related information, 544
headings, 543-544
HTML, 51, 391-394
liquid, 395
measurements, 408
re-ordering sections, 398-399
redesigning, 400-401
style sheet, 394-398
tables, 390-391, 408

layout style sheets, writing, 394-401

left property (CSS positioning), 196

<legend> tag, 351

length units (CSS), 180-181

Lesson 1 Exercises, 23

Lesson 1 Quiz, 22

Lesson 2 Exercises, 48

Lesson 2 Quiz, 48

Lesson 3 Exercises, 65-66

Lesson 3 Quiz, 65

Lesson 4 Exercises, 80

Lesson 4 Quiz, 79

Lesson 5 Exercise, 98

Lesson 5 Quiz, 97

Lesson 6 Exercises, 130

Lesson 6 Quiz, 130

Lesson 7 Exercise, 172

Lesson 7 Quiz, 172

Lesson 8 Exercise, 209

Lesson 8 Quiz, 208-209

Lesson 9 Exercise, 258

Lesson 9 Quiz, 257-258

Lesson 10 Exercise, 310

Lesson 10 Quiz, 309-310

Lesson 11 Exercise, 353

Lesson 11 Quiz, 352

Lesson 12 Exercise, 388

Lesson 12 Quiz, 387-388

Lesson 13 Exercise, 410
Lesson 13 Quiz, 409
Lesson 14 Exercise, 434
Lesson 14 Quiz, 433-434
Lesson 15 Exercise, 457
Lesson 15 Quiz, 457
Lesson 16 Exercise, 487
Lesson 16 Quiz, 486
Lesson 17 Exercise, 530
Lesson 17 Quiz, 529-530
Lesson 18 Exercise, 566
Lesson 18 Quiz, 565
Lesson 19 Exercise, 592
Lesson 19 Quiz, 591-592
Lesson 20 Exercise, 618
Lesson 20 Quiz, 617-618
Lesson 21 Exercise, 655
Lesson 21 Quiz, 654-655
Lesson 22 Exercise, 689
Lesson 22 Quiz, 689
** tag, 82, 86-87, 95**
libraries (JavaScript), 459-460
Dojo, 461
Google Web Toolkit, 462
jQuery, 461-463
adding/removing content, 474-477
AJAX, 480-484
binding events, 465
enabling users, 485
hiding and showing elements, 466-467
manipulating attributes, 473-474
manipulating classes, 468-471
manipulating form values, 471-473
modifying page styles, 466-468
modifying web page content, 468-477
retrieving style sheet properties, 467-468
sample script, 463-464
slowly loading pages, 485
special effects, 478-480
Midori, 462
MochiKit, 462
MooTools, 462
Prototype, 461-462
Yahoo, UI (YUI), 461
line breaks (tables), setting, 278-280
line feeds, 601
linear organization, websites, 35-36
linear with alternatives organization, websites, 36-37
linear/hierarchical organization, websites, 38-39
link attribute (body tag), 237-238
link checkers, 695
link menus, 539
creating, 110-112, 161-162
<link> tag, 127, 175
linked windows, 491
accessibility, 587
base tag, 496-498
creating, 491-496, 499-500
frameset tag, 499-500
cols attribute, 500-501
rows attribute, 501-502
JavaScript, opening with, 523-526
links, 127-129
active state, 206
anchors, 113
creating, 113-114
hash signs and, 119
linking in same page, 120
sample exercises, 115-119
colors, 555
creating, 100
sample exercise, 101-105
<a> tags, 100-101
CSS, modifying with, 206-207
definitions as links, 551
explicit navigation links, 550
"here" links, 549-550
home page links, 558
hover state, 206
images, 228-231
implicit navigation links, 550
link menus
creating, 110-112
design tips, 546-547
local pages, pathnames, 105-107
navigation links, 541
remote pages, linking to, 108-110
text links, design tips, 547-548
troubleshooting, 604
web pages, adding to, 452-454
when to use, 550, 552
writing accessible HTML, 583-584
Links Web browser, 18, 694
Linode, 597
liquid layouts, 395
list-style property, 96
list-style-image properties, unordered lists, 89
list-style-image property, 96
list-style-position properties, unordered lists, 89
list-style-position property, 96
list-style-type property, 96
listings
13.1 (Using div Tags to Create Sections for Positioning), 392-393
13.2 (Style Sheet for Page Layout), 395
13.3 (Style Sheet for Colors and Fonts), 397-398

13.4 (Moving One Section Before Another), 399
13.5 (Float-Based Layouts in CSS), 401-402
13.6 (Randomly Organized Style Sheet), 405-406
13.7 (Better-Organized Style Sheet), 406-407
lists, 82-83
customizing, numbered lists, 85-87
glossary lists, 90
HTML tags for, 82-83
importance of, 539
nesting lists, 92-93
ordered lists, 83-87
other uses for, 94-95
unordered lists, 87-89
***Little Brother*, 11-12**
LiveCounter, 693
loading external data, AJAX, 481-484
local pages, linking, pathnames, 105-107
location object (JavaScript), 429
log file parsers, 696
log files, 613
Google Analytics, 614-616
logical style tags, 132-134
longdesc attribute (frame tag), 506
longdesc attribute (frameset tag), 528
longdesc attribute (img tag), 585
loop attribute (audio tag), 384
loop attribute (video tag), 367
loops
for loops, 423
iterations, 422
JavaScript, 422-423
PHP, 630
controlling loop execution, 632-633
for, 631
foreach, 630-631
while and do…while loops, 632
while loops, 423-424
Lynx, 18, 694
formatted text, 135
<legend> tag, grouping controls, 340-341

M

magazines, 29
magic target names, 520
Mailto URLs, 21, 125-126
main topics, determining, 31-32
management, file management, 594
manipulating classes, jQuery, 468-471
map tag, 248-249
Mapedit program, 246, 695
marginheight attribute (frame tag), 506
marginheight attribute (frameset tag), 527
marginheight attribute (iframe tag), 521, 528
margins
CSS box model, 185-189
frame margins, 506
marginwidth attribute (frame tag), 506
marginwidth attribute (frameset tag), 527
marginwidth attribute (iframe tag), 521, 528
marketing. *See* advertising
markup languages, 53
maxlength attribute (<input> tag), 322-323
measurement units (CSS), 180-181
measurements, 408
media attribute, <link> tag, 175
media types (Internet), 372
MediaTemple.com, 700
MediaWiki, 662, 674-677, 687
downloading and installing, 675-676
MediaWiki.org, 699
menus
creating with <option> and <select> tags, 332-334
link menus
creating, 110-112
design tips, 546-547
messages, alert messages, displaying, 438
meta tags, 611
method attribute (<form> tag), 313-314, 319
get method, 313-314
get/post methods, 319-320
post method, 313-314
methods
DOM, accessing, 454
getElementsByClassName(), 460
Microsoft ASP.NET, 653
Microsoft Bing, 610
Microsoft Internet Explorer, 9, 15-16
document type definition (DTD), 188
Microsoft Internet Information Server, 697
Microsoft Silverlight, 698
middle alignment (images), 220
Midori JavaScript library, 462
MIME types list website, 697
Mint website, 696
minus sign (-) subtraction operator, 418
mm unit (CSS), 181
MochiKit JavaScript library, 462

modulus operator (%), 418
MooTools JavaScript library, 462
Mosaic, 9
Movable Type, 699
movies, embedding, SWFObject, 376-377
moving files betwen Web servers, 600
carriage returns/line feeds, 601
filename restrictions, 601
FTP (File Transfer Protocol), 601-603
Moving One Section Before Another listing (13.4), 399
Mozilla Developer Center, 694
Mozilla Firefox, 9, 17, 693
MSDN (Microsoft Developer's Network) Online, 694
multimedia
embedding, embed tag, 375
web pages, scaling down, 590
multiple attribute, <option> tag, 334
multiplication operator (*), 418
MySpace, advertising websites via, 606
MySQL, 666
myString function (JavaScript), 424

N

name attribute (<a> tag), 114
name attribute (<frame> tag), 506
name attribute (<frameset> tag), 528
name attribute (<iframe> tag), 521, 528
name attribute (<map> tag), 248
name attribute (<object> tag), 373
named entities, 150
named frames, 517
names
domain names, registering, 596
filenames, 599-601
frame names, 506, 517
naming
colors, 234-235
frames, 517
naming conventions, CSS classes and IDs, 208
navigation
adding to web pages, 572-573
breadcrumbnavigation, 572
DOM, node properties, 450, 341-342
navigation icons, 229-231
navigation links, 541
nested tags, 70
nesting
framesets, 516
tables, 309
nesting lists, 92-93
nesting tags, 105
Network Solutions website, 596
networks
intranet, 26
intranets, 9
newsgroups, accessing, 19
newspapers (online), 12-13
Ning.com, 45
<nobr> tag, 169
node properties, DOM, navigating, 450
noembed tag, 375
noframes tag, 503-504, 519-520, 528
non-anonymous FTP, URLs, 125
noresize attribute (frame tag), 506
noresize attribute (frameset tag), 527
noresize attribute (iframe tag), 521
noshade attributes
without values, 145
nowrap attribute (<td> tag), 279-280, 307
nowrap attribute (<th> tag), 279-280, 307
number of pages, determining, 558-561
numbered entities, 150
numbered lists
customizing, 85-87
HTML tags for, 83
NYPL Online Style Guide, 697

O

object tag, 386
alternative content, 374
attributes, 373
embedding Flash multimedia, 370-374
video tag, using with, 382
object-oriented PHP, 652
objects
JavaScript, 427-428
thisform, 438
Ogg Theora container format, 362
converting to, 366
versus H.264 codec, 387
** tag, 82-83, 86, 95**
onblur event handler, 430
onchange event handler, 430
onclick event handler, 430
onfocus event handler, 430
online books, 11
online documentation, 29

online newspapers, 12-13
onload event handler, 430
onmouseover event handler, 430
onselect event handler, 430
onsubmit event handler, 430, 437
onunload event handler, 430
opening HTML tags, 56
Opera, 18, 693
operators
JavaScript, 417-418
string concatenation operators, 627
opinion gathering websites, 29
optimizing search engines, 611
<option> tag, 351
ordered lists, 83
customizing, 84-87
numbered lists, 83
organizing
HTML documents, 598
default index files, 599-600
directories, 599
filenames, 599-600
Web server setup, 598-599
style sheets, 404-406
websites, 539-540
<option> tag
menus, creating, 332-334
multiple attribute, 334
selected attribute, 334

P

<p> tag, 169, 385
packaged software, 664
deploying applications, 666
relational databases, 664-666
padding
CSS box model, 185-189
page descriptions, creating, 160
page layout, 543
absolute positioning, 394-396
colors/fonts, 397-398
consistent layout, 545
CSS, positioning, 196-205
fixed, 395
floated columns, 401-402
group-related information, 544
headings, 543-544
HTML, 51, 391-394
liquid, 395
measurements, 408
re-ordering sections, 398-399
redesigning, 400-401
style sheet, 394-398
tables, 390-391, 408
page-level styles (CSS), creating, 174
pages. *See* web page
Pair.com, 700
paragraphs, HTML tags for, 75
param tag, 386
parameters, JavaScript, 424
parent target name, 520
parsing HTML documents, 51
password attribute, <input> tag, 350
password controls, creating with <input> tag, 323-324
password protection, 595
passwords, forms for, creating, 312-315, 317
pathnames
absolute pathnames, 106-107
local pages, linking, 105-107
relative pathnames, 107
releative pathnames, 105-106
paying for search placement, 612
PBworks.com, 45
pc unit (CSS), 181
percent sign (%)
email addresses, 126
modulus operator, 418
URL escape codes, 122
percentage units (CSS), 181
percentages
measurements, 408
widths, specifying with, 270
permissions, checking, 604
personal information, websites, 28
photograph hosting, Flickr, 682-684
Photoshop, 212
PHP (Hypertext Preprocessor), 619-622, 653-654
arrays, 624-626
built-in functions, 633
comments, 622-623
conditional operators, 629-630
conditional statements, 628-629
cookies, 652
database connectivity, 651
expanding knowledge of, 650-651
file uploads, 652
files, deploying and editing, 654
includes, 647-650
loops, 630
controlling execution, 632-633
for, 631
foreach, 630-631
while and do…while loops, 632
object-oriented PHP, 652
processing forms, 636-637

parameters with multiple values, 637-638
presenting forms, 642-647
validating forms, 638-642
regular expressions, 651
running on your computer, 621-622
scripts, browser dependence, 654
sending email, 652
server-side scripting, 698
sessions, 652
strings, 626-628
user-defined functions, 634
returning values, 635
variables, 623-624

PHP and MySQL Web Development, 650

PHP interpreter, 621

physical style tags, 135-136

Pilgrim, Mark, 589

pixels, measurements, 408

planning pages. See also content, 157

plug-ins, 10

Plug-ins page, WordPress, 673

plug-ins page attribute (embed tag), 375

plus sign (+) addition operator, 418

PNG (Portable Network Graphics), 214

polls, websites, 29

polygons, imagemap coordinates, 246-247

pop-up windows

JavaScript, opening with, 523-526
setting up, 524

popup() function, 524-525

port numbers, URLs, 122

Portable Network Graphics (PNG), 214

positioning (CSS), 196-197

absolute positioning, 199-201
relative positioning, 197-198
static positioning, 196
top/left/bottom/right properties, 196
z-index property (stacking), 202-205

post method (<form> tag), 313-314, 319-320

Posterous.com, 44, 699

PostgreSQL, 666

posting pages, WordPress, 672

pound signs (#)

CSS and, 178
numbered entities, 150

<pre> tag, 139-141, 168

preformatted text, 139-141

preload attribute (audio tag), 384

preload attribute (video tag), 367

preparing images for Web, 212

presenting forms, PHP, 642-647

preventing cross-site scripting, 636-637

processing forms (PHP), 636-637

parameters with multiple values, 637-638
presenting forms, 642-647
validating forms, 638-642

programs

form validation script, 436
checkform() function, 437
code listing, 439-443
onsubmit event handler, 437
selected variable, 438
thisform object, 438
Mapedit, 246

progressive enhancement, web pages, 532

promoting websites. See advertising websites

proofreading, 543

properties

CSS (cascading style sheets)
background-attachment, 239
background-position, 239
color, 236
font-family, 170
font-size, 170
font-style, 170
font-variant, 170
font-weight, 170
text-align, 170
text-decoration, 170
list-style, 96
list-style-image, 96
list-style-position, 96
list-style-type, 96
style sheets, retrieving, 467-468

protocols

FTP (File Transfer Protocol), 601-603

Prototype JavaScript library, 461-462

pt unit (CSS), 181

publishing websites, 29

file organization, 598
default index files, 599-600
directories, 599
filenames, 599-600
Web server setup, 598-599
moving files between Web servers, 600
carriage returns/line feeds, 601
case sensitivity, 601
filename restrictions, 601

FTP (File Transfer Protocol), 601-603
online books, 11
Web servers, 594
authentication, 595
choosing, 595-597
file management, 594
file types, 594
media types, 594
security, 595
server-side processing, 595
server-side scripts and forms processing, 594
px unit (CSS), 181

Q

QuickTime, 697
container format, 362
QuirksMode.org, 695-696
quizzes
Lesson 1, 22
Lesson 2, 48
Lesson 3, 65
Lesson 4, 79
Lesson 5, 97
Lesson 6, 130
Lesson 7, 172
Lesson 8, 208-209
Lesson 9, 257-258
Lesson 10, 309-310
Lesson 11, 352
Lesson 12, 387-388
Lesson 13, 409
Lesson 14, 433-434
Lesson 15, 457
Lesson 16, 486
Lesson 17, 529-530
Lesson 18, 565
Lesson 19, 591-592
Lesson 20, 617-618
Lesson 21, 654-655
Lesson 22, 689
quotation HTML tags. See <blockquote> tag, 147
quotation marks (“), href attributes, 103

R

radio attribute, <input> tag, 351
radio buttons, creating with <input> tag, 326-327
Randomly Organized Style Sheet listing (13.6), 405-406
readonly form controls, creating, 342-343
RealNetworks, 697
rectangles, imagemap coordinates, 248
reducing image file sizes, 553-554
Register.com, 597
registering domain names, 596
registration form example (form controls), 334, 337-339
registration forms, 14
regular expressions, PHP, 651
rel attribute, <link> tag, 175
related_posts() function, 674
relational databases, 664-666
relative pathnames, 105-106
linking local pages, 105-107
versus absolute pathnames, 107
relative positioning (CSS), 197-198
relative units (CSS), 180-181
remote pages, linking to, 108-110
reports, Google Analytics, 615-616
require_once, 649-650
reserved characters (character entities), 152
reserved words, JavaScript, 419
reset attribute, <input> tag, 351
reset buttons, creating with <input> tag, 325
retrieving style sheet properties, JQuery, 467-468
reverse scaling, images, 232
right alignment (images), 223-224
right property (CSS positioning), 196
rows (tables), 262
see also *cells*
alignment, 300-302
creating, 262-264
grouping, 300-302
rows attribute (frameset tag), 501-502, 514-515, 527
rowspan attribute, 308
rowspan attribute (<td> tag), 288
rowspan attribute (<th> tag), 288
Ruby on Rails, 653
rules attribute, 307

S

<s> tag, 135, 168
Safari, 693
Safari (Apple), 9, 18
<samp> tag, 168
as logical style tag, 133
samples
creating and formatting pages, 157
adding attributes, 166-167
adding content, 158-159
adding tables of contents, 159

Bookworm web page code, 163-165
frameworks, 157-158
link menus, 161-162
page descriptions, 160
planning the page, 157
signatures, 163
testing results, 165-166
unordered lists, 162-163
HTML documents, creating, 59, 76-77
link menus, creating, 110-112
linking anchors, 115-119
linking remote pages, 108-110
links, creating, 101, 103-105
scripts, JQuery, 463-464
tables
creating, 266-269
service specification table, 290-294, 296
***Sams Teach Yourself PHP, MySQL, and Apache All in One*, 650**
sans-serif fonts, 62
scaling images, 232
scanning web pages, 539-540
school servers, 595
SCP (Secure Copy), 46
screen magnifiers, 581
screen readers, 581
<script> tag, JavaScript, 415
script.aculo.us, 696
scripts
cross-site scripting, preventing, 636-637
JavaScript
code listing, 439-443
documentation, 432
for loops, 423
form validation, 436-438
if statement, 422
while loops, 423-424
PHP, browser dependence, 654
servers-side scripts and forms processing, 594
scrolling frames, 506-507
scrolling attribute (<frame> tag), 506-507
scrolling attribute (<frameset> tag), 527
scrolling attribute (<iframe> tag), 521, 528
Search Engine Watch website, 609
search engines, 609
adding to web pages, 569-570
Ask.com, 610
Bing, 610
choosing, 617
Google, 609-610
optimization, 611
placement, paying for, 612
Search Engine Watch website, 609
site rankings, 610
Yahoo, 610
Section 508, 580-581
Secure Copy (SCP), 46
Secure FTP (SFTP), 46
Secure Socket Layer (SSL), 595
security
authentication, 595
file permissions, checking, 604
forms, 343
SSL (Secure Socket Layer), 595
web aplications, 688
Web servers, 595
<select> tag, 332-334, 351
selected attribute, <option> tag, 334
selected variable, 438
selectors (CSS), 176
classes, 177-178
contextual selectors, 176-177
IDs, 177-178
self target name, 520
semicolons, named entities, 150
sending email, PHP, 652
serifs, 62
server administration websites, 697
server efficiency, JavaScript, 413
server-side file processing, 595
server-side images, 243-244
server-side scripting, websites, 698
server-side scripts and forms processing, 594
servers (Web), 594
see also *web servers*
authentication, 595
choosing, 595
bandwidth limitations, 598
domain parking, 597
IPPs (Internet presence providers), 596
ISPs (Internet service providers), 596
personal servers, 597
school servers, 595
Web presence providers, 596
work servers, 595
file management, 594
file types, 594
media types, 594
moving files between, 600
carriage returns/line feeds, 601
filename restrictions, 601
FTP (File Transfer Protocol), 601-603
security, 595

server-side file processing, 595
server-side scripts and forms processing, 594
troubleshooting
file access, 603
server access, 603
web servers, 26
sessions (PHP), 652
SFTP (Secure FTP), 46
SGML (Standard Generalized Markup Language), 50
shape attribute (area tag), 248
shapes, imagemap coordinates, 246-248
shopping catalogs, websites, 29
showing elements, JavaScript, 443-451
signatures
creating, 163
designing, 561-562
web pages, 564
Silverlight, 698
simple spiders, 695
site indexes, 609
Ask.com, 610
Bing, 610
choosing, 617
Google, 609-610
optimization, 611
site rankings, 610
Yahoo, 610
site management packages, storyboarding with, 44
Site Meter, 693
site-wide style sheets, 407
sites. ***See*** **websites**
sitewide style sheets (CSS), creating, 175-176
Sixapart.com, 699
size attribute (<hr> tags), 143
size attribute (<input> tag), 322-323
size attribute (<select> tag), 333
sizes
CSS box model, controlling, 189-192
images, 232, 553-554
tables, width, 269
Skitch.com, 699
slash (/), division operator, 418
Slicehost, 597
<small> tag, 135, 168
Smashing Magazine, 694
social media, advertising websites on, 606-609
software, packaged software, 664
deploying applications, 666
relational databases, 664-666
sound, embedding, 383-385
<embed> tag, 375
source element, <video> tag, 369-370
<source> tag, 386
spacing
images, 226-228
table cells, 274-275
spam, 687
** tag, 137, 168**
span attribute (<colgroup> tag), 296, 308
special characters (URLs), 122
special effects, web pages, 478-480
spell checking, 543
splitting topics across pages, 558
src attribute (audio tag), 384
src attribute (embed tag), 375
src attribute (frame tag), 507
src attribute (frameset tag), 527
src attribute (iframe tag), 521, 528
src attribute (img tag), 215, 255
src attribute (<script> tag), 415
src attribute (video tag), 367
SSL (Secure Socket Layer), 595
stacking (CSS), 202-205
standalone web page, designing, 541
Standard Generalized Markup Language (SGML), 50
standards (CSS), 208
standards compliance, websites, 532-538
standby attribute (object tag), 373
StatCounter, 693
statements, JavaScript, 421-423
static positioning (CSS), 196
stopping text wrapping, 225-226
storyboarding, 48
Strict specification (XHTML), 534
string concatenation operators, 627
strings (PHP), 626-628
strip_tags() function, 644
** tag, 132, 168**
structures (websites), 33, 68
hierachical/linear, 38-39
hierarchical, 33-35
HTML, writing with, 391-393
ideas for, 33
linear, 35-36
linear with alternatives, 36-37
web, 39-42
style attribute, 62
alternatives to, 174
<style> tag, 174
style guides (HTML), 697

Style Sheet for Colors and Fonts listing (13.3), 397-398
Style Sheet for Page Layout listing (13.2), 395
style sheets (CSS), 173
creating, 61-62
HTML tags, combining with, 62
laying out, 394-398
properties
background-attachment, 239
background-position, 239
color, 236
retrieving, 467-468
web design
organization, 404-406
site-wide, 407
web designs, 403-404
styles
CSS (cascading style sheets), 236-239
web pages, JQuery, 466-468
<sub> tag, 135, 168
submit attribute (<input> tag), 350
submit buttons (forms), 315
creating with <input> tag, 324
images as submit buttons, 327-328
subtraction operator (-), 418
summary attribute (<table> tag), 261-262, 303, 582
<sup> tag, 135, 168
SWFObject, 384-385
Flash movies, embedding, 376-377
syntax, JavaScript, 415-428

T

tabindex attribute (<a> tag), 123, 341, 584
<table> tag, 261, 305
align attribute, 282
bgcolor attribute, 280
border attribute, 270-273
cellpadding attribute, 273-274
cellspacing attribute, 274
char attribute, 303
charoff attribute, 303
summary attribute, 261-262, 303, 582
width attribute, 269-270
tables, 259-260, 304
alignment, 282-283
captions, 286-287
cells, 283-286
rows, 300-302
body of, defining, 301
borders, changing, 270-273
captions, 260
alignment, 286-287
creating, 265-269
cells, 260
alignment, 283, 285-286
creating, 262-264
empty cells, 264-265
padding, 273-274
spacing, 274-275
spanning multiple rows/columns, 287-289
colors, background colors, changing, 280-282
columns
aligning, 296-299
grouping, 296-299
width of, 275, 277
creating, 260
sample exercise, 266-267, 269
<table> tag, 261-262
footers, 301
headings, 260, 300-301
laying out, 390-391
layout, 408
line breaks, setting, 278-280
nesting, 309
rows
alignment, 300-302
creating, 262-264
grouping, 300-302
sample exercises, service specification table, 290-296
troubleshooting, 309
when to use, 303
whitespace, eliminating, 275
width of, 269
writing accessible HTML, 582-583
tables of contents, adding to sample pages, 159
tags (HTML), 51, 68, 78, 95
<a>
accesskey attribute, 584
event handlers, 123
HTML 4.01 additions to, 123
links, creating, 100-101
name attributes and, 114
tabindex attribute, 123, 584
target attribute, 491-493, 496, 520
title attribute, 583
<abbr>, 133
accessibility, 587
<acronym>, 134
<address>, 147, 168, 561
<area>, 248-249
alt attribute, 249
coords attribute, 249
href attribute, 249
shape attribute, 248
<audio> tag, 383-384, 386
<b>, 135, 168
<base>, 496-498, 527
<big>, 135, 168

<blockquote>, 147-148, 168
<body>, 69-70
alink attribute, 237-238
bgcolor attribute, 236
link attribute, 237-238
text attribute, 236, 238
vlink attribute, 237-238

, 169, 226
clear attribute, 226
table line breaks, 278
brackets, 56
<button>, 351
form controls, 330-331
<caption>, 265-269, 305
align attribute, 286
case sensitivity, 56
, 170
character entities, 149
encoding, 151
named entities, 150
numbered entities, 150
reserved characters, 152
character-level elements, 132
character-level elements versus block-level elements, 132
<cite>, 168
as logical style tags, 133
versus <blockquote>, 147
closing, 56, 70, 82
<code>, 168
as logical style tags, 133
<col>, 297-298, 305
<colgroup>, 296-298, 305
comments, 75-76, 79
conventions, founding of, 52
CSS (cascading style sheets), combining with, 62
<dd>, 82, 91, 96
designing web pages, 254-255
<dfn>, 168
as logical style tags, 133
<dir>, 82
, 153-155, 170
<dl>, 91, 96
DOCTYPE identifiers, 68
<dt>, 82, 91, 96
, 116, 168
as logical style tags, 132
<embed>, 375-376, 386
<fieldset>, 351
<form>, 350
<frame>, 502-503, 527-529
bordercolor attribute, 505-506
longdesc attribute, 506
marginheight attribute, 506
marginwidth attribute, 506
name attribute, 506
noresize attribute, 506
scrolling attribute, 506-507
src attribute, 507
<frameset>, 499-500, 527-529
cols attribute, 500-501, 514-515
nesting, 516
rows attribute, 501-502, 514-515
<head>, 69
heading levels, 72, 74
style attributes and, 62
<hr>, 142, 143, 169
attributes, 143-145
<html>, 68-69
<i>, 116, 135, 168
<iframe>, 521-523, 528
<img>, 214-215, 255-256
align attribute, 220-224
alt attribute, 215-216, 584
border attribute, 229
height attribute, 232
hspace attribute, 226-227
longdesc attribute, 585
src attribute, 215
usemap attribute, 249-250
vspace attribute, 226-227
width attribute, 232
indented code, 73
<input>, 350-351
<kbd>, 168
as logical style tags, 133
<label>, 314, 351
displaying label elements, 320-321
<legend>, 351
, 82, 86-87, 95
<link>, 175
<lists>, 82-83
glossary lists, 90
nesting lists, 92-93
numbered lists, 83
numbered lists, customizing, 85-87
unordered lists, 88
unordered lists, customizing, 88-89
logical style tags, 132-134
<map>, 248-249
name attribute, 248
markup languages, 53
meta tags, 611
nested, 70, 105
<nobr>, 169
<noembed>, 375
<noframes>, 503-504, 519-520, 528
<object>, 370-374, 382, 386
, 82-83, 95
type attributes and, 86
opening, 56
<option>, 351
origins, 53-54
<p>, 169, 385
<paragraph>, 75

<param>, 386
physical style tags, 135-136
<pre>, 139-141, 168
<s>, 135, 168
<samp>, 133, 168
sample pages, creating, 76-77, 157
adding attributes, 166-167
adding content, 158-159
adding tables of contents, 159
Bookworm web page code, 163-165
frameworks, 157-158
link menus, 161-162
page descriptions, 160
planning the page, 157
signatures, 163
testing results, 165-166
unordered lists, 162-163
<script>, JavaScript, 414-415
<select>, 351
<small>, 135, 168
<source>, 386
, 137, 168
, 132, 168
<style>, type attribute, 174
<sub>, 135, 168
<sup>, 135, 168
<table>261, 305
align attribute, 282
border attribute, 270-273
cellpadding attribute, 274
cellspacing attribute, 274
summary attribute, 261-262, 582
width attribute, 269-270
<tbody>, 301, 305
colspan attribute, 288
nowrap attribute, 279
rowspan attribute, 288
valign attribute, 284
<td>, 262, 305
<textarea>, 331-332, 351
text alignment, 153
blocks of elements, 153, 155
individual elements, 153
<tfoot>, 301, 305
<th>, 262, 288, 305, 582
<thead>, 300
<thread>, 305
<title>, 70-71
<tr>, 262, 305
<tt>, 135, 168
<u>, 135, 168
, 82, 95
<var>, 168
as logical style tags, 133
<video>, 366-370, 382, 386
<wbr>, 169
XHTML 1.0 and, 55
target attribute (<a> tag), 491-493, 496, 520
target attribute (<base> tag), 497, 527
target names, 520
<tbody> tag, 301, 305
<td> tag, 262-264, 305
colspan attribute, 288
nowrap attribute, 279-280
rowspan attribute, 288
valign attribute, 284
***Teach Yourself PHP, MySQL, and Apache All in One*, 650**
templating systems, content management, 663
terms (glossary lists). *See* <dt> tag
testing
forms, 352
required fields, 438-439
results, 165-166
text, 541
aligning images with, 220-222
character formatting, design tips, 541
colors, 236-237
proofreading, 543
spell checking, 543
substituting for images, 215-216
text links
colors, 555
design tips, 547-548
"here" links, 549-550
when to use, 550, 552
wrapping around images, 223-224, 226
text alignment, 153
blocks of elements, 153-155
individual elements, 153
text attribute (<input> tag), 350
text attribute (<body> tag), 236-238
text controls, creating with <input> tag, 322-323
text editors, 57
ASCII format, 57
HTML editors, 62-64
HTML-Kit, 57
TextWrangler, 57
WYSIWYG, 63
text form controls, 314
text formatting, HTML pages, 60
text wrapping, stopping, 225-226
text-align property, align attribute, 287
text-align property (CSS), 170
text-decoration properties (CSS), 137-139, 170
<textarea> tag, 351
form controls, 331-332
texttop alignment (images), 221

TextWrangler text editor, 57
<tfoot> tag, 301, 305
<th> tag, 262-264, 305, 582
colspan attribute, 288
nowrap attribute, 279-280
rowspan attribute, 288
valign attribute, 284
this argument, 432
thisform object, 438
<thead> tag, 301, 305
tiling images, 238
Tim Berners-Lee's style guide, 697
title attribute (<link> tag), 175
title attribute (<a> tag), 583
<title> tag, 70-71
titles, 541
Tomcat web servers, 697
tools
blogging tools, 660-661
HTML Tidy, 538
W3C Validator, 535-537
top alignment (images), 220
top levels, 107
top property (CSS positioning), 196
top target name, 520
topics, determining, 31-32
TopStyle HTML editor, 695
<tr> tag, 262-264, 305
Transitional specification (XHTML), 534
Transmit, 602
troubleshooting web pages
client-side imagemaps, 257
CSS, 207
file access, 603
file display errors, 605
images, 604
links, 604
tables, 309
Web server access, 603
<tt> tag, 135, 168
Tumblr.com, 44, 699
Twitter
accounts, creating, 607
advertising websites via, 607
content, embedding, 684-685
type attribute (<embed> tag), 375
type attribute (<Input> tag), 314, 322-323, 350
button, 328
checkbox, 325
file, 329
hidden, 329
radio, 326
type attribute (<link> tag), 175
type attribute (object tag), 373
type attribute (style tag), 174
TypePad.com, 44, 661, 667-669, 699

U

<u> tag, 135, 168
** tag, 82, 87-88, 95**
underscore (_), 491
Unicode, 151
uniform resource locators (URLs). *See* URLs (uniform resource locators)
units of measure (CSS), 180-181
UNIX, conventions of, 106
unordered lists, 87-88
creating, 162-163
customizing, 88-89
unset() function, 625
updates, web applications, automatic applications, 688
uploading videos, YouTube, 358-359
URLs (uniform resource locators), 11, 20-21, 120-123, 127-129
adding to web pages, 570-572
anonymous FTP, 124-125
CSS, 175, 182
directories, 122
escape codes, 122
file, 126-127
FTP (File Transfer Protocol), 19
hostnames, 121-122
HTTP (Hypertext Transfer Protocol), 19, 123-124
mailto, 21, 125-126
newsgroups, 19
non-anonymous FTP, 125
protocols, 121
special characters, 122
specifications websites, 698
usemap attribute (object tag), 373
usemap attribute (img tag), 249-250
Usenet newsgroups, accessing, 19
user experience levels, designing web pages for, 569
user preferences, 568
designing web pages, 568-575
user-defined functions, PHP, 634-635
Using div Tags to Create Sections for Positioning listing (13.1), 392-393

V

validating
accessibility, 588-589
web pages
HTML Tidy, 538
W3C Validator, 535-537
validating forms
JavaScript, 436

checkform() function, 437
code listing, 439-443
onsubmit event handler, 437
selected variable, 438
thisform object, 438
PHP, 638-642
validation function, JavaScript, 437
validators, 695
Cynthia Says, 588-589
valign attribute (<td> tag), 284, 308
valign attribute (<th> tag), 284, 308
value attribute (<input> tag), 87, 315, 323, 325
values
em, 409
forms, manipulating, 471-473
<var> tag, 133, 168
variables, 635
JavaScript, 418-420
PHP, 623-624
selected, 438
vertical frames, 500-501
combining with horizontal frames, 514-515
video
container formats, 362-363
customizing, YouTube, 359-360
embedding, 356-361
embed tag, 375
Flash, 370-374
SWFObject, 376-377
video tag, 366-370
H.264 video codec, 362-366
hosting, 361-366
YouTube, 357
Ogg Theora container format, 362
converting to, 366
uploading, YouTube, 358-359
web pages, embedding, 356-361
video players (Flash), 378
Flowplayer, 380-381
JW Player, 378-380
<video> tag, 386
attributes, 367
embedding video, 366-370
object tag, using with, 382
source element, 369-370
View Source functionalities, 76
Vimeo, 356, 360, 386
visual styles, HTML tags, evolution of, 52
VLC website, 698
vlink attribute (<body> tag), 237-238
vspace attribute (<iframe> tag), 521
vspace attribute (<img> tag), 226-227, 256
vspace attribute (<object> tag), 373

W

W3 Validator, 535-537
W3C (World Wide Web Consortium), 22, 54, 694
Web Accessibility Initiative, 589
W3C validator, 695
W3Schools.com, 694
<wbr> tag, 169
WDG HTML validator, 695
web, 8
control of, 21
cross-platform compatibility, 10
decentralized nature of, 10-11
dynamic nature of, 11-12
graphical navigation, 9
hypertext, 8-9
interactivity, 13-14
W3C (World Wide Web Consortium), 21-22, 54, 694
web applications
automatic updates, 688
installing, 688
security, 688
web browsers, 15
accessibility, alternative browsers, 581
accessing services, 18
as HTML formatters, 51
audio browsers, HTML tags for, 133-134
Camino, 18
compatibility, 387
exclusive content development, 15
FTP servers, navigating, 125
Google Chrome, 18
home pages, 26
image coordinates, 251
JavaScript
compatibility, 433
integration, 413
Konqueror, 18
Links, 18
Lynx, 18, 135
Microsoft Internet Explorer, 16
Mosaic, 9
Mozilla Firefox, 17
multiple browser usage, advantages, 16
Opera, 18
PHP scripts, dependence, 654
plug-ins, 10
purpose of, 15
Safari, 18

websites, 693
standards compliance, 532-538
web design, style sheets, 403-407
web documents. *See* web pages
web hosting, 44
choosing, 595-597
commercial, 596
content-management applications, 44-45
free, 597
providers, 700
setting up, 45-46
web applications, installing, 688
web pages, 26
accessibility
Building Accessible Websites (italic), 589
Dive Into Accessibility website, 589
validating, 588-589
W3C Web Accessibility Initiative, 589
audio, embedding, 383-385
colors, 234
background colors, 236
naming, 234-235
text colors, 236
content, adding to, 452-455
creating, progressive enhancement, 532
design, 538
accessibility, 579
backgrounds, 555
brevity, 539
browser-specific terminology, 542
clarity, 539
consistent layout, 545
determining user preferences, 573-575
emphasis, 541, 587
for first-time users versus regular users, 573
frames, 570
grouping related information, 544
hardware considerations, 555
headings, 543-544
images, 254-255, 552-554
links, 546-552, 555, 558
navigation, 572-573
number of pages, 558-561
organizing for quick scanning, 539-540
page signatures, 561-562
page validation, 535-538
proofreading, 543
search engines, 569-570
spell checking, 543
splitting topics across pages, 558
standalone pages, 541
standards compliance, 532-538
URLs, 570-572
user experience levels, 569
user preferences, 568-569
XHTML, 579
document types, choosing, 534-535
documents, converting to, 564
elements, hiding/showing, 443-451
Flash multimedia, embedding, 376-377
forms, validating with JavaScript, 436-440, 442-443
frames, 489-491
borders, 504-506
browser support for, 490
detriments, 527
frame tag, 502-507, 527
horizontal frames, 501-502
inline frames, 521-523
linking documents to, 502-503, 518
magic target names, 520
margins, 506
names, 506
naming, 517
noframes tag, 503-504, 519-520, 528
scrolling, 506-507
vertical frames, 500-501
framesets, 491
base tag, 496-498
combining vertical and horizontal frames, 514-515
content pages, 507-514
creating, 491-496, 499-500
frameset documents, 498-499
frameset tag, 499-502
horizontal frames, 501-502
nesting, 516
vertical frames, 500-501
Halloween House example, 216-218
images
adding, 214-218
aligning with text, 220-222
alternative text, 215-216
as links, 228-231
background images, 238-240
borders, 233
bullets, 243

GIF (Graphics Interchange Format), 213, 256
Halloween House web page example, 217-218
height/width, 232
image etiquette, 254-255
inline images, 214-215, 219
JPEG (Joint Photographic Experts Group), 213, 256
navigation icons, 229-231
PNG (Portable Network Graphics), 214
preparing for Web, 212
scaling, 232
spacing around, 226-228
wrapping text around, 223-226
integrating, JavaScript, 456
linking. See links, 108
links
adding to, 452-454
colors, 555
definitions as links, 551
explicit navigation links, 550
here' links, 549-550
home page links, 558
implicit navigation links, 550
link menus, 546-547
text links, 547-548
when to use, 550-552
modifying content and styles, JQuery, 466-477
multimedia, scaling down, 590
number of, 558, 560-561
page signatures, 561-562
remote pages, 108
signatures, 564
slowly loading, JavaScript, 485
special effects, JQuery, 478-480
text formatting, design tips, 541
URLs, 120-123
validating
HTML Tidy, 538
W3 Validator, 535-537
video
embedding, 356-361, 366-370
hosting, 361-366
web presence providers, 596
web publishing tools, websites, 699
web servers, 11, 19-20, 26, 594
Apache, 697
authentication, 595
choosing, 595
bandwidth limitations, 598
domain parking, 597
IPPs (Internet presence providers), 596
ISPs (Internet service providers), 596
personal servers, 597
school servers, 595
Web presence providers, 596
work servers, 595
directories, index files, 599
file management, 594
file types, 594
FTP servers, navigating, 125
media types, 594
Microsoft Internet Information Server, 697
moving files between, 600
carriage returns/line feeds, 601
filename restrictions, 601
FTP (File Transfer Protocol), 601-603
security, 595
server-side file processing, 595
server-side scripts and forms processing, 594
Tomcat, 697
troubleshooting, 603
websites, 11, 26, 48
accessibility, 588-591
Dive Into Accessibility website, 589
Adobe, 212
advertising, 605
brochures, 612
business cards, 612
links from other sites, 606
social media, 606-609
analytic, 693
anatomy of, 26-27
Ask.com, 610
AWStats, 696
Bing, 610
Building Accessible Websites, 589
content
goal setting, 30-31
hierarchical, 33-35
hierarchical/linear, 38-39
ideas, 28-30
linear, 35-36
linear with alternatives, 36-37
main topics, 31-32
navigation, 32-42
organization, 32-42
storyboarding, 42-44
web, 39-42
CuteFTP, 602
design, 538
backgrounds, 555
brevity, 539

browser-specific terminology, 542
clarity, 539
consistent layout, 545
emphasis, 541
grouping related information, 544
hardware considerations, 555
headings, 543-544
images, 552-554
links, 546
standards compliance, 532-538
colors, 555
definitions as links, 551
explicit navigation links, 550
external files, 591
Fetch, 602
forms, 13-14
validating with JavaScript, 436-443
FTP Explorer, 602
Google, 609-610
here links, 549-550
home pages, 26, 558
hits, 605
imagemaps, 695
implicit navigation links, 550
initial visit, 27
IrfanView, 212
JavaScript, 696
link menus, 546-547
links
colors, 555
definitions as links, 551
explicit navigation links, 550
here' links, 549-550
home page links, 558
implicit navigation links, 550
link menus, 546-547
text links, 547-548
when to use, 550, 552
Links browser, 694
log file parsers, 696
log files, 613
Network Solutions, 596
Opera, 693
page validation
HTML Tidy, 538
W3 Validator, 535-537
publishing
file organization, 598-600
moving files between Web servers, 600-603
server-side scripts and forms processing, 594
Web servers, 594-597
Web serversMost Web servers provide the ability to restrict access, 595
Search Engine Watch, 609
search engines, 609
Ask.com, 610
Bing, 610
Google, 609-610
Search Engine Watch website, 609
Yahoo, 610
Section 508, 581
Transmit, 602
troubleshooting, 603
file access, 603
file display errors, 605
images, 604
links, 604
server access, 603
URLs, 120-123
W3C Validator, 535
W3C Web Accessibility Initiative, 589
web hosting, 44
content-management applications, 44-45
setting up, 45-46
web pages, 26
web structure, 39-42
wget, 693
when to use, 550, 552
number of pages, 558, 560-561
organizing for quick scanning, 539-540
page signatures, 561-562
page validation, 535-538
proofreading, 543
spell checking, 543
splitting topics across pages, 558
standalone pages, 541
text links, 547-548
Yahoo, 610
web structure, websites, 39-42
Webalizer website, 696
weblogs. *See* blogs
WebM container format, 363
webmasters, 598
webservers, 697
wget, 693
wget website, 693
while and do…while loops, PHP, 632
while loops, 423
JavaScript, 423-424
white space, <button> tag, 331
whitespace, tables, 275
width
columns (tables), percentages as, 270
images, 232
table columns, 275-277
tables, 269
width attribute (<embed> tag), 375
width attribute (<hr> tag), 144
width attribute (<iframe> tag), 521, 528

width attribute (<img> tag), 232
width attribute (<object> tag), 373
width attribute (<table> tag), 269-270, 307-308
width attribute (<video> tag), 367
width attribute (<colgroup> tag), 297
wikidot.com, 45
Wikipedia.org, 40-42, 661
wikis, 661-662
MediaWiki, 674-677
downloading and installing, 675-676
WikiWikiWeb, 662
WinAmp, 697
window object (JavaScript), 429
windows, linked, 491
base tag, 496-498
creating, 491-496, 499-500
frameset tag, 499-502
Windows Media, 698
WordPress, 658, 661, 669-674
confirmation file, 670
directories, 670
installation page, 671
one-click installation, 670
Plug-ins page, 673
posting page, 672
WordPress.com, 44, 699
work related content, websites, 28
work servers, 595
workflow systems, content management, 663
World Wide Web Consortium (W3C), 22
wrapping text around images, 223-226
writing accessible HTML, 582-585
WYSIWYG (what you see is what you get)
text editors, 63
versus HTML, 50

X, Y, Z

XAMPP, 621
XHTML, 54
versus HTML, 534-535
XHTML 1.0 Frameset specification, 534
XHTML 1.0 Strict specification, 534
XHTML 1.0 Transitional specification, 534
XHTML 1.0, 579, 590
attributes without values, 145

 tags, converting, 146
HTML tags and, 55
website, 698
XHTML 1.0 Frameset, designing web pages, 579
XHTML 1.0 Strict, designing web pages, 579
XHTML 1.0 Transitional, designing web pages, 579
XML (eXtensible Markup Language), 54
HTML5, 578
Yahoo, 610
Developer Network, 694
JavaScript library, 461
navigation, 572
Web Analytics, 693
Yale HyperText style guide, 697
yellow_page window, 492-496
YouTube, 356, 385-386, 682
customizing videos on, 359-360
embedding video, 356-361
hosting videos on, 357
uploading videos to, 358-359
YUI Compressor, 696
YUI Library, 696
z-index property (CSS), 202-205

Try Safari Books Online FREE

Get online access to 5,000+ Books and Videos

Safari Books Online

FREE TRIAL—GET STARTED TODAY!
www.informit.com/safaritrial

Find trusted answers, fast

Only Safari lets you search across thousands of best-selling books from the top technology publishers, including Addison-Wesley Professional, Cisco Press, O'Reilly, Prentice Hall, Que, and Sams.

Master the latest tools and techniques

In addition to gaining access to an incredible inventory of technical books, Safari's extensive collection of video tutorials lets you learn from the leading video training experts.

WAIT, THERE'S MORE!

Keep your competitive edge

With Rough Cuts, get access to the developing manuscript and be among the first to learn the newest technologies.

Stay current with emerging technologies

Short Cuts and Quick Reference Sheets are short, concise, focused content created to get you up-to-speed quickly on new and cutting-edge technologies.

Addison Wesley · Adobe Press · Alpha · Cisco Press · FT Press · IBM Press · lynda.com · Microsoft Press · New Riders · O'Reilly · Peachpit Press · Prentice Hall · Que · Redbooks · Sams · SAS Publishing · Sun Microsystems · Wharton School Publishing · Wiley

informIT.com THE TRUSTED TECHNOLOGY LEARNING SOURCE

PEARSON

InformIT is a brand of Pearson and the online presence for the world's leading technology publishers. It's your source for reliable and qualified content and knowledge, providing access to the top brands, authors, and contributors from the tech community.

Addison-Wesley Cisco Press EXAM/CRAM IBM Press. QUE PRENTICE HALL SAMS | Safari Books Online

LearnIT at InformIT

Looking for a book, eBook, or training video on a new technology? Seeking timely and relevant information and tutorials? Looking for expert opinions, advice, and tips? **InformIT has the solution.**

- Learn about new releases and special promotions by subscribing to a wide variety of newsletters.
  Visit **informit.com/newsletters**.
- Access FREE podcasts from experts at **informit.com/podcasts**.
- Read the latest author articles and sample chapters at **informit.com/articles**.
- Access thousands of books and videos in the Safari Books Online digital library at **safari.informit.com**.
- Get tips from expert blogs at **informit.com/blogs**.

Visit **informit.com/learn** to discover all the ways you can access the hottest technology content.

Are You Part of the IT Crowd?

Connect with Pearson authors and editors via RSS feeds, Facebook, Twitter, YouTube, and more! Visit **informit.com/socialconnect**.

informIT.com THE TRUSTED TECHNOLOGY LEARNING SOURCE PEARSON

Addison-Wesley Cisco Press EXAM/CRAM IBM Press. QUE PRENTICE HALL SAMS | Safari Books Online

THIS PRODUCT

informit.com/register

Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to **informit.com/register** to sign in or create an account. You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:

- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

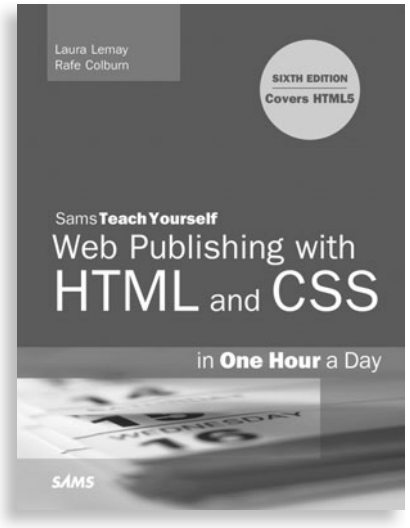
About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you're looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.

informIT.com
THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison-Wesley | Cisco Press | Exam Cram
IBM Press | Que | Prentice Hall | Sams

SAFARI BOOKS ONLINE



FREE Online Edition

Your purchase of **Sams Teach Yourself Web Publishing with HTML** includes access to a free online edition for 45 days through the Safari Books Online subscription service. Nearly every Sams book is available online through Safari Books Online, along with more than 5,000 other technical books and videos from publishers such as Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, O'Reilly, Prentice Hall, and Que.

SAFARI BOOKS ONLINE allows you to search for a specific answer, cut and paste code, download chapters, and stay current with emerging technologies.

Activate your FREE Online Edition at www.informit.com/safarifree

- **STEP 1:** Enter the coupon code: ZATIQVH.
- **STEP 2:** New Safari users, complete the brief registration form. Safari subscribers, just log in.

If you have difficulty registering on Safari or accessing the online edition, please e-mail customer-service@safaribooksonline.com

Safari Books Online

Addison Wesley · Adobe Press · Alpha · Cisco Press · FT Press · IBM Press · lynda.com · Microsoft Press · New Riders · O'Reilly · Peachpit Press · Prentice Hall · Que · Redbooks · Sams · SAS Publishing · Sun microsystems · Wharton School Publishing · Wiley