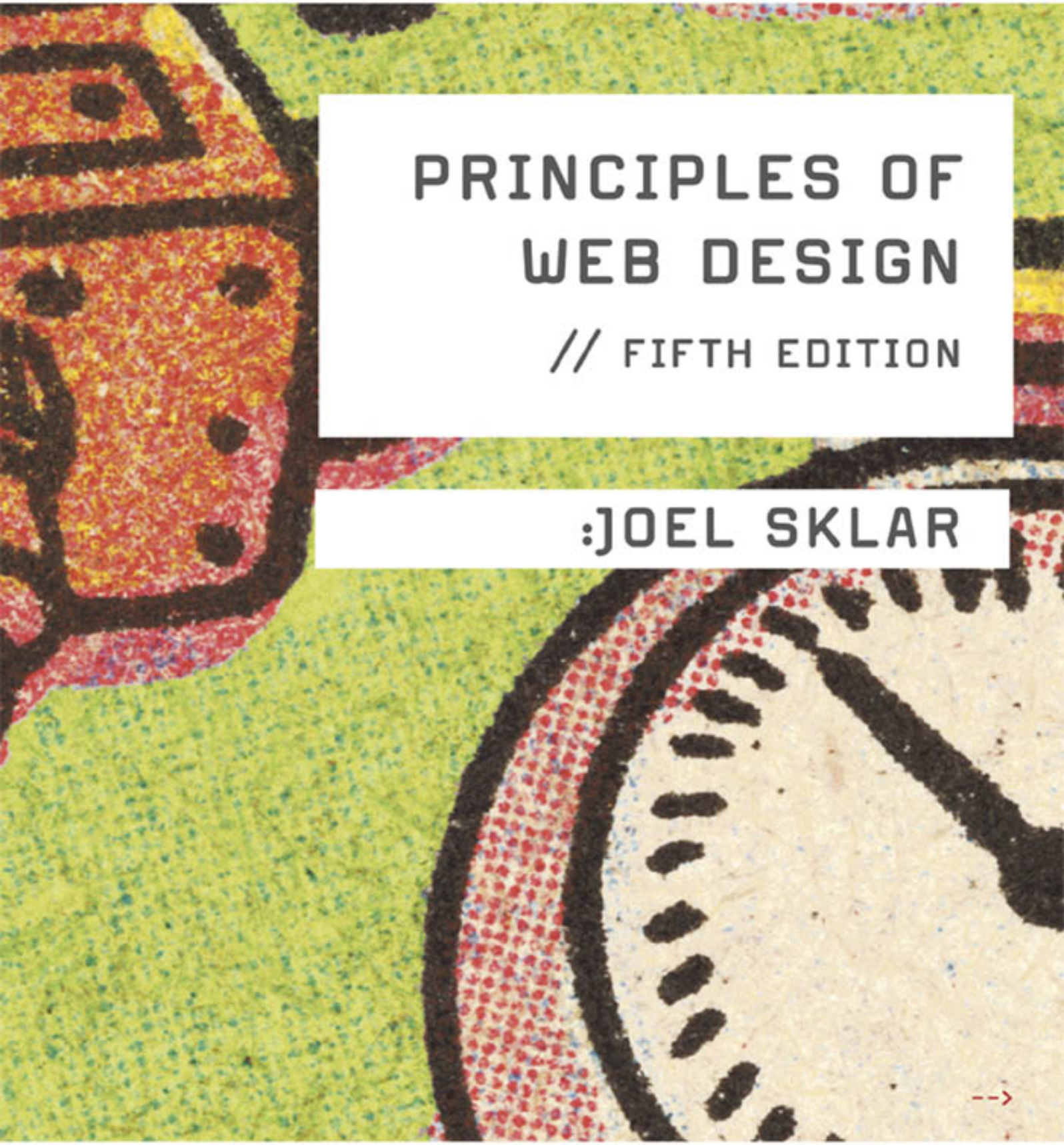
<!-- THE WEB TECHNOLOGIES SERIES



PRINCIPLES OF WEB DESIGN

// FIFTH EDITION

:JOEL SKLAR

-->

This is an electronic version of the print textbook. Due to electronic rights restrictions, some third party content may be suppressed. Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. The publisher reserves the right to remove content from this title at any time if subsequent rights restrictions require it. For valuable information on pricing, previous editions, changes to current editions, and alternate formats, please visit www.cengage.com/highered to search by ISBN#, author, title, or keyword for materials in your areas of interest.

PRINCIPLES OF WEB DESIGN

This page intentionally left blank

FIFTH EDITION

PRINCIPLES OF WEB DESIGN

JOEL SKLAR

COURSE TECHNOLOGY
CENGAGE Learning™

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

COURSE TECHNOLOGY
CENGAGE Learning™

Principles of Web Design, Fifth Edition
Joel Sklar

Executive Editor: Marie Lee

Acquisitions Editor: Brandi Shailer

Senior Product Manager: Alyssa Pratt

Editorial Assistant: Stephanie Lorenz

Senior Content Project Manager: Catherine DiMassa

Developmental Editor: Lisa Ruffolo

Associate Marketing Manager: Shanna Shelton

Art Director: Faith Brosnan

Print Buyer: Julio Esperas

Cover Designer: Cabbage Design Company

Cover Image: © istockphoto/Shaun Martin

Compositor: Integra

© 2012 Course Technology, Cengage Learning

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product, submit all requests online at **www.cengage.com/permissions**
Further permissions questions can be emailed to
permissionrequest@cengage.com

Library of Congress Control Number: 2010942347

ISBN-13: 978-1-111-52870-6
ISBN-10: 1-111-52870-5

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Some of the product names and company names used in this book have been used for identification purposes only and may be trademarks or registered trademarks of their respective manufacturers and sellers.

Any fictional data related to persons or companies or URLs used throughout this book is intended for instructional purposes only. At the time this book was printed, any such data was fictional and not belonging to any real persons or companies.

Course Technology, a part of Cengage Learning, reserves the right to revise this publication and make changes from time to time in its content without notice.

The programs in this book are for instructional purposes only.

They have been tested with care, but are not guaranteed for any particular intent beyond educational purposes. The author and the publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs.

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil and Japan. Locate your local office at: **www.cengage.com/global**

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Course Technology, visit **www.cengage.com/coursetechnology**

Purchase any of our products at your local college store or at our preferred online store **www.cengagebrain.com**

Printed in the United States of America
1 2 3 4 5 6 7 16 15 14 13 12 11 10

Brief Contents

	Preface	xviii
CHAPTER 1	HTML and the Modern Web	1
CHAPTER 2	Web Site Design Principles	48
CHAPTER 3	Site Planning	105
CHAPTER 4	Cascading Style Sheets	150
CHAPTER 5	Web Typography	191
CHAPTER 6	Box Properties	244
CHAPTER 7	Page Layouts	293
CHAPTER 8	Graphics and Color	337
CHAPTER 9	Site Navigation	399
CHAPTER 10	Data Tables	447
CHAPTER 11	Web Forms	481
CHAPTER 12	Web Page Design Studio	526

APPENDIX A HTML5 Reference 593

APPENDIX B CSS Reference 614

APPENDIX C CSS Media Queries 622

APPENDIX D Print Style Sheets 627

Index 632

Contents

Preface xviii

CHAPTER 1 HTML and the Modern Web 1

Creating Web Pages with HTML 2
Structure of a Basic Web Page 3
HTML in the Browser 5
Adding Style with CSS 5
Organizing Information with Hypertext 8
The History of HTML 8
A Need for Standards 9
XML and XHTML: A New Direction 11
Problems with XHTML 13
A Proposal for HTML5 13
Working with HTML5 14
HTML5 Loose and Strict Syntaxes 16
Choosing the Correct Syntax 17
Choosing the Correct Document Type and MIME type . . . 18
Creating Syntactically Correct Code 19
Element Categories 21
New Elements in HTML5 23
Attributes in HTML5 25
Obsolete Elements in HTML5 25
Using HTML5 Elements for Page Structure 25
Interactive Capabilities in HTML5 28
Choosing an HTML Editor 29
Using Good Coding Practices 30
Stick to the Standards 31
Use Semantic Markup 31
Validate Your Code 32
Migrating from Legacy HTML to HTML5 32
Chapter Summary 33
Key Terms 34
Review Questions 36

Hands-On Projects 37
Individual Case Project 45
Team Case Project 46

CHAPTER 2 Web Site Design Principles 48

Understanding the Web Design Environment 49
Browser Compatibility Issues 50
Connection Speed Differences 51
Browser Cache and Download Time 52
Device and Operating System Issues 53
Designing for Multiple Screen Resolutions 54
Wide-Screen Displays 55
Handheld Devices 57
Flexible Page Layouts 59
Fixed-Width Page Layouts 63
Suggestions for Solving the Screen Resolution Dilemma . .65
Crafting the Look and Feel of the Site 66
Balance Design and Content 66
Plan for Easy Access to Your Information 67
Plan for Clear Presentation of Your Information 68
Creating a Unified Site Design 69
Plan Smooth Transitions 70
Use a Grid to Provide Visual Structure 72
Use Active White Space 75
Designing for the User 77
Design for Interaction 80
Design for Location 82
Keep a Flat Hierarchy 87
Use Hypertext Linking Effectively 88
How Much Content Is Too Much? 90
Reformat Content for Online Presentation 91
Designing for Accessibility 92
WCAG 2.0 Guidelines 94
Chapter Summary 98
Key Terms 99
Review Questions 100
Hands-On Projects 101
Individual Case Project 103
Team Case Project 103

CHAPTER 3 Site Planning 105

Understanding the Web Site Development Process 106
Requirements and Specification 107
Information Design and Taxonomy Creation 108

Graphic Design and Page Template Creation 108
Construction and Content Development. 110
Quality Assurance and User Testing 110
Publishing and Promotion 111
Ongoing Maintenance. 111
Creating a Site Specification 111
Identifying the Content Goal. 112
Analyzing Your Audience 115
Using Web Analytics 117
Identifying Technology Issues and Accessibility Constraints 119
Identifying Software Tools. 120
Building a Web Site Development Team 121
Creating Conventions for Filenames and URLs 122
Naming Files. 123
Using Complete or Partial URLs 125
Setting a Directory Structure 125
Using a Single Folder Structure 126
Using a Hierarchical Folder Structure. 127
Creating a Site Storyboard 128
Organizing the Information Structure 129
Publishing Your Web Site 135
Choosing a Web Hosting Service Provider 135
Registering a Domain Name 137
Web Hosting Service Comparison Checklist 138
Uploading Files with the File Transfer Protocol 138
Testing Your Web Site 140
Testing Considerations 140
Usability Testing 141
Chapter Summary 142
Key Terms. 144
Review Questions 145
Hands-On Projects 147
Individual Case Project 148
Team Case Project. 149

CHAPTER 4 Cascading Style Sheets **150**

Recognizing the Benefits of Using CSS. 151
The Evolution of CSS 151
CSS Style Rules 152
Writing Clean CSS Code. 156
Activity: Building a Basic Style Sheet 157
Using Inheritance to Write Simpler Style Rules 159
Examining Basic Selection Techniques 161

Using Type Selectors . . . 161
Grouping Selectors . . . 162
Combining Declarations . . . 162
Using Descendant Selectors . . . 162
Using the Universal Selector . . . 163
Activity: Applying Basic Selection Techniques . . . 163
Using Class and ID Selectors . . . 166
Using the Class Selector . . . 167
Using the id Selector . . . 169
Using the and Elements . . . 170
Working with Elements . . . 170
Working with Elements . . . 171
Using Other Selectors . . . 172
Using Attribute Selectors . . . 172
Using Pseudo-Class and Pseudo-Element Selectors . . . 173
Understanding How the Cascade Affects Style Rules . . . 180
CSS3 Selectors . . . 181
Chapter Summary . . . 184
Key Terms . . . 184
Review Questions and Exercises . . . 186
Hands-On Projects . . . 187
Individual Case Project . . . 189
Team Case Project . . . 190

CHAPTER 5 Web Typography . . . 191

Understanding Type Design Principles . . . 192
Choose Fewer Fonts and Sizes . . . 192
Use Common Web Fonts . . . 193
Specifying Proprietary Web Fonts . . . 195
Design for Legibility . . . 196
Avoid Creating Text as Graphics . . . 197
Understanding CSS Measurement Units . . . 197
Absolute Units . . . 198
Relative Units . . . 199
CSS Property Descriptions . . . 201
Using the CSS Font Properties . . . 202
Specifying Font Family . . . 202
Specifying Font-Face . . . 205
Specifying Font Size . . . 206
Specifying Font Style . . . 208
Specifying Font Variant . . . 208
Specifying Font Weight . . . 209
Specifying Font Stretch . . . 210
Using the Font Shortcut Property . . . 210

Using the CSS Text Spacing Properties 211
Specifying Text Indents 212
Specifying Text Alignment 213
Specifying Line Height 214
Specifying Vertical Alignment 215
Specifying Letter Spacing 217
Specifying Word Spacing 218
Specifying Text Decoration 219
Specifying Capitalization 220
Specifying Text Shadow 221
Currently Unsupported CSS3 Properties 222
Activity: Building a Font and Text Properties Style Sheet . . . 224
Adding the <style> Section 225
Styling the Headings 225
Styling the Paragraphs 226
Making the Second-Level Headings More Distinctive . . . 227
Capitalizing Key Words 228
Customizing Bulleted and Numbered Lists 230
Specifying the list-style-type Property 231
Specifying the list-style-image Property 233
Specifying the list-style-position Property 234
Using the list-style Shorthand Property 235
Chapter Summary 236
Key Terms 237
Review Questions And Exercises 238
Hands-On Projects 239
Individual Case Project 243
Team Case Project 243

CHAPTER 6 Box Properties 244

Understanding the CSS Visual Formatting Model 245
Specifying the Display Type 247
Using the CSS Box Model 248
Measurement Values 251
Applying the Margin Properties 251
Specifying Margins 251
Negative Margins 253
Collapsing Margins 254
Zeroing Margins 254
Applying the Padding Properties 255
Padding Property Shorthand Notation 257
Applying the Border Properties 258
Specifying Border Style 259
Specifying Border Width 262

Specifying Border Color 264
Using the Border Shorthand Properties 266
Specifying Rounded Borders 267
Using the Page Layout Box Properties 268
Setting Element Width 269
Setting Element Height 272
Floating Elements 272
Clearing Elements 275
Controlling Overflow 276
Creating Box Shadows 278
Activity: Creating a Simple Page Layout 280
Chapter Summary 286
Key Terms 287
Review Questions and Exercises 287
Hands-On Projects 288
Individual Case Project 291
Team Case Project 291

CHAPTER 7 Page Layouts 293

Understanding the Normal Flow of Elements 294
Using the Division Element to Create Content Containers . . 296
Creating Floating Layouts 298
Solution 1: Using a Normal Flow Element 300
Solution 2: Using the Clear Property 302
Floating Elements Within Floats 303
Fixing Column Drops 305
Clearing Problem Floats 306
Building a Flexible Page Layout 307
Controlling Flexible Layout Width 310
Activity: Creating a Flexible Layout 310
Building a Fixed Page Layout 316
Controlling Fixed Layout Centering 318
Activity: Creating a Fixed Layout 319
Chapter Summary 330
Key Terms 330
Review Questions 331
Hands-On Projects 332
Individual Case Project 335
Team Case Project 336

CHAPTER 8 Graphics and Color 337

Understanding Graphics File Formats 338
GIF 338

JPG . . . 341
PNG . . . 342
SVG . . . 342
Using Interlacing and Progressive Display . . . 343
Where You Can Find Images . . . 344
Choosing the Right Format . . . 345
Choosing a Graphics Tool . . . 345
Using the Image Element . . . 346
Replacing Image Element Attributes with Style Sheet Properties . . . 347
Specifying alt and title Attribute Text . . . 348
Specifying Image Width and Height . . . 349
Sizing Graphics for the Page . . . 352
Controlling Image Properties with CSS . . . 353
Removing the Hypertext Border from an Image . . . 353
Aligning Text and Images . . . 354
Floating Images . . . 355
Adding White Space Around Images . . . 356
Understanding Computer Color Basics . . . 358
Color Depth . . . 358
Dithering . . . 358
Using the Web Palette . . . 359
Creating Web Site Color Schemes . . . 359
Using Color Wisely . . . 364
Specifying CSS Color Values . . . 365
Understanding Element Layers . . . 367
Controlling Color Properties with CSS . . . 368
Specifying Color Values . . . 368
Setting the Default Text Color . . . 369
Changing Link Colors . . . 370
Specifying Background Color . . . 370
Setting the Page Background Color . . . 372
Creating a Text Reverse . . . 373
Controlling Background Images with CSS . . . 373
Specifying a Background Image . . . 374
Creating a Page Background . . . 375
Specifying Background Repeat . . . 377
Creating a Vertical Repeat . . . 377
Creating a Horizontal Repeat . . . 378
Creating a Nonrepeating Background Image . . . 379
Specifying Background Position . . . 380
Positioning Repeating Background Images . . . 382
Chapter Summary . . . 383
Key Terms . . . 384

Review Questions 385
Hands-On Projects 387
Individual Case Project 397
Team Case Project 397

CHAPTER 9 Site Navigation 399

Creating Usable Navigation 400
Locating the User 401
Limiting Information Overload 402
Activity: Building Navigation Structures 402
Linking with a Text Navigation Bar 404
Linking to Chapter Pages 407
Adding Internal Linking 409
Adding a Page Navigation Bar 410
Linking to External Document Fragments 413
Adding Page Turners 416
Adding Contextual Linking 418
Navigation Summary 419
Using Graphics for Navigation and Linking 420
Using the alt Attribute 420
Using Icons for Navigation 421
Using Lists for Navigation 422
Removing Default Padding and Margin 423
Removing Default Bullets 423
Building Horizontal Navigation Bars 424
Customizing the Horizontal Navigation Bar 425
Controlling Navigation Bar Width 427
Controlling Navigation Button Width 428
Building Vertical Navigation Bars 429
Using Background Color and Graphics to Enhance Navigation 432
Indicating History 432
Indicating Location 433
Creating Hover Rollovers 435
Changing Text Color and Background Color on Hover . . 435
Changing Background Images on Hover 436
Underlining on Hover 437
Chapter Summary 438
Key Terms 439
Review Questions 439
Hands-On Projects 440
Individual Case Project 445
Team Case Project 446

CHAPTER 10 Data Tables **447**

Using Table Elements 448
Collapsing Table Borders 451
Spanning Columns 451
Spanning Rows 452
Using Table Headers and Footers 454
Grouping Columns 455
Styling the Caption 457
Styling Table Borders 458
Applying Padding, Margins, and Floats to Tables 461
Using Padding 461
Using Margins and Floats 463
Styling Table Background Colors 465
Specifying Background Color 465
Creating Alternate Color Rows 466
Creating Background Hover Effects 467
Activity: Applying Table Styles 469
Chapter Summary 474
Key Terms 475
Review Questions 475
Hands-On Projects 476
Individual Case Project 480
Team Case Project 480

CHAPTER 11 Web Forms **481**

Understanding How Forms Work 482
Using the <form> Element 484
Using get or post 484
Using the mailto Action 485
Creating Input Objects 485
Labeling Form Elements 486
Creating Text Boxes 488
Creating Check Boxes 489
Creating Radio Buttons 490
Creating Submit and Reset Buttons 491
Creating a Password Entry Field 494
Using the <select> Element 494
Using the <textarea> Element 497
Creating Input Groupings 498
Styling Forms with CSS 500
Aligning Form Elements 501
Styling Fieldset and Legend Elements 504

Activity: Building a Form 507
Adding Check Boxes 509
Adding a List Box and Radio Buttons 512
Adding Submit and Reset Buttons 514
Styling the Labels 515
Styling the Fieldsets and Legends 517
Chapter Summary 519
Key Terms. 520
Review Questions 521
Hands-On Projects 521
Individual Case Project 525
Team Case Project 525

CHAPTER 12 Web Page Design Studio 526

The Initial Design Process 527
Creating the Mock-up Web Page 533
Examining the HTML Code. 536
Setting Page Background Color and Default Font 540
Creating the Page Wrapper Division 540
Creating the Page Header 542
Styling the Header Text 544
Creating the Navigation Bar 546
Creating the Column Divisions. 552
Creating the Left Column 553
Creating the Main Column 555
Creating the Right Column. 558
Creating the Footer 560
Building the Small Feature Boxes 561
Styling the Small Feature Box Content 566
Building the Feature Article Section 570
Styling the Search and Links Section 573
Building the Links Section 577
Finished Page Code 583
Chapter Summary 589
Review Questions 590
Hands-On Projects 591
Individual and Team Case Project 591

APPENDIX A HTML5 Reference 593

Alphabetical HTML5 Reference 594
Obsolete Elements 607

Global attributes 607
Event Attributes 609
Numeric and Character Entities 609

APPENDIX B CSS Reference 614

CSS Notation Reference 615
Alphabetical CSS Property Reference 615
CSS Measurement Units 621

APPENDIX C CSS Media Queries 622

Overview of CSS Media Queries 623
Media Query Syntax 623
Applying Media Queries 623
Supported Media Types. 624
Media Features 624
Examples of CSS Media Features 626

APPENDIX D Print Style Sheets 627

Applying Print Styles 628
Creating Print Styles 629
Specifying Fonts and Color 629
Specifying Background Colors 629
Removing Elements. 629

Index 632

Preface

P*rinciples of Web Design, Fifth Edition*, leads you through the entire Web site creation process, from start to finish, while developing and enhancing your HTML, CSS, and visual design skills along the way. You will learn how to create accessible Web sites that let users easily and quickly navigate through your information, regardless of browser type, connection speed, or browsing device. Whether you are building a site from scratch or redesigning an existing site, the principles presented in this text will help you deliver your Web content in a more interesting, accessible, and visually exciting way.

This edition reflects the latest in Web design trends with expanded sections, plenty of new content, and a reorganized topic flow. The examples and activities emphasize building standards-based Web designs using the latest Web technologies including HTML5 and CSS3. You will examine current Web design theories and view a variety of Web sites, learning to focus on both the user's needs and the requirements of the content you want to deliver. You will learn about the Web project design process and see how to take an initial rough sketch and turn it into a finished layout. Through hands-on activities you will gain experience controlling all Web design aspects including typography, color, backgrounds, page layout, and navigation.

Principles of Web Design, Fifth Edition is printed in full color, allowing you to better see how designers use color in the example Web sites that illustrate Web design principles. Updated illustrations and screen shots throughout the book reflect current browser, device, and Web design trends.

The Intended Audience

Principles of Web Design, Fifth Edition is intended for anyone who has a working knowledge of HTML and wants to apply those skills to design attractive, informative Web pages. To work effectively

with the content of this book, you need to understand the basics of HTML at the code level. You may have taken an introductory class in HTML, or taught yourself HTML with the help of a book or the Web. You should be able to build a simple Web page that includes text, hyperlinks, and graphics. Additionally, you should be comfortable working with computers and know your way around your operating system, whether Windows, Macintosh, or Linux.

The Approach

As you progress through the book, you practice the design techniques by studying the supplied coding samples, looking at the example pages and Web sites, and applying the principles to your own work. Each chapter concludes with a summary, individual and team project ideas, and a review section that highlights and reinforces the major concepts of each chapter. To complete a case project, you should complete each chapter in sequence.

Overview of This Book

The examples and exercises in this book will help you achieve the following objectives:

- Apply your HTML skills to building designed Web pages
- Learn about the latest release of HTML, called HTML5, and see how it can adapt to a variety of Web design needs
- Understand the effects of browser and different device types on your design choices
- Learn to build portable, accessible Web sites that clearly present information
- Gain a critical eye for evaluating Web site design
- Effectively use graphics, typography, and color in your work
- Build user-focused navigation to help your users find content easily
- Use CSS layout techniques to build fixed or flexible page layouts

In **Chapter 1** you will explore how HTML is used along with CSS to create Web pages, learn about the new elements and capabilities of HTML5, how to choose the best syntax for the Web pages you are going to create, and how to create correct code.

Chapter 2 covers the design principles that you will apply to your Web page design as you work through the book. You will look at a variety of Web sites and learn to focus on both the user's needs and information requirements of your site. In **Chapter 3** you will learn about the Web development project lifecycle and the importance of planning your Web site before you start coding. You will also learn about important file naming and directory conventions, as well as creating a flowchart that depicts the information structure of your site. You will learn how to publish your site to the Web and plan for ongoing site maintenance and updates. **Chapter 4** introduces CSS, including its basic syntax and selection techniques, and explains how to control style information in a single file or across and entire Web site. **Chapter 5** explains how you can use CSS as a potent style language to manipulate a variety of text properties to achieve professional, effective typographic design.

Chapter 6 introduces the CSS visual formatting model and the box model, which control the way content is displayed on a Web page. You also explore the CSS box properties to set the margin, padding, and border characteristics of block-level elements and to enhance the display of content in the browser. **Chapter 7** expands on the concepts introduced in Chapter 6, and demonstrates how to use CSS layout techniques to create multicolumn Web pages that can either be flexible based on the browser size and screen resolution, or fixed to a definite width. **Chapter 8** explains the effective use of images and color on your Web site, including image file formats, correct use of the <img> element, and computer color basics. **Chapter 9** focuses on navigation and how to help your users find content easily, know where they are at all times, and see where they can go within your Web site. **Chapter 10** discusses how to create attractive legible data tables, using CSS. In **Chapter 11**, you will learn how to work with HTML form elements to build interactive Web pages that collect information from a user and process it on the Web server. Finally, in **Chapter 12**, you will apply a wide variety of skills you learned in the book to building a mock-up layout design for a fictional Web site.

Features

Principles of Web Design, Fifth Edition contains many teaching aids to assist the student's learning.

CHAPTER OBJECTIVES Each chapter in this book begins with a list of the important concepts to be mastered within the chapter. This list provides you with a quick reference to the contents of the chapter as well as a useful study aid.

ILLUSTRATIONS, TABLES, AND SCREEN SHOTS Illustrations help you visualize common components and relationships. Tables list conceptual items and examples in a visual and readable format. Updated screen shots reflect the latest technology being used in Web design.

NOTES Chapters contain Notes designed to provide you with practical advice and proven strategies related to the concept being discussed.

COVERS MODERN WEB DESIGN TECHNIQUES All new content on HTML5 and CSS-based layouts demonstrate the latest methods for creating Web pages using standards-based design techniques.

FULL COLOR WEB PAGE ILLUSTRATIONS Web page figures and other illustrations are shown in full color so you can assess how color affects Web page content and how designers use color effectively in sample Web sites.

CHAPTER SUMMARIES Each chapter's text is followed by a summary of chapter concepts. These summaries provide a helpful way to recap and revisit the ideas covered in each chapter.

KEY TERMS Each chapter includes a list of newly introduced vocabulary. The list of key terms provides a mini-review of the major concepts in the chapter.

REVIEW QUESTIONS End-of-chapter assessment begins with a set of approximately 15 to 20 review questions that reinforce the main ideas introduced in each chapter. These questions ensure that you have mastered the concepts and have understood the information you have learned. Some questions have been updated for the Fifth Edition.

HANDS-ON PROJECTS Although it is important to understand the concepts behind Web design topics, no amount of theory can improve real-world experience. To this end, along with conceptual explanations, each chapter provides Hands-On Projects related to each major topic aimed at providing you with practical experience. Some of these include researching information from people, printed resources, and the Internet, as well as installing and using some of the technologies discussed. Because the Hands-On Projects ask you to go beyond the boundaries of the text itself, they provide you with practice implementing Web design skills in real-world situations. Many projects have been updated for the Fifth Edition.

CASE PROJECTS The individual and team case projects at the end of each chapter are designed to help you apply what you have learned to business situations much like those you can expect to encounter as a Web designer. Depending on instructor preferences, you can work on your own or with a team to independently synthesize and evaluate information, examine potential solutions, and make recommendations, much as you would in an actual design situation. These have also been updated for the Fifth Edition.

Online Companion

The online companion to accompany *Principles of Web Design* has been an important component to this book since the First Edition. For the Fifth Edition, it offers greater enhancement to textbook learning by providing updated information and Web links for further research. The URL for this site is *www.joelsklar.com/pwd5*.

Instructor Resources

The following supplemental materials are available when this book is used in a classroom setting. Most of the teaching tools available with this book are provided to the instructor on a single CD-ROM.

ELECTRONIC INSTRUCTOR'S MANUAL The Instructor's Manual that accompanies this textbook includes additional instructional material to assist in class preparation, including Sample Syllabi, Chapter Outlines, Technical Notes, Lecture Notes, Quick Quizzes, Teaching Tips, Discussion Topics, and Key Terms.

EXAMVIEW® This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. The computer-based and Internet testing components allow students to take exams at their computers, and also save the instructor time by grading each exam automatically.

POWERPOINT PRESENTATIONS Microsoft PowerPoint slides are available for each chapter. These are included as a teaching aid for classroom presentation, to make available to students on

the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.

DATA FILES Files that contain all of the data necessary for the Hands-On Projects and Case Projects are provided for students at www.cengagebrain.com, and are also available on the Instructor Resources CD-ROM.

SOLUTION FILES Solutions to end-of-chapter Review Questions, Hands-On Projects, and Case Projects are provided on the Instructor Resources CD-ROM and may also be found on the Cengage Learning Web site at *login.cengage.com*. The solutions are password protected.

DISTANCE LEARNING Course Technology is proud to present online test banks in WebCT and Blackboard, to provide the most complete and dynamic learning experience possible. Instructors are encouraged to make the most of your course, both online and offline. For more information on how to access your online test bank, contact your local Course Technology Cengage Learning sales representative.

Read This Before You Begin

The following information will help you as you prepare to use this textbook.

To the User of the Data Files

To complete the steps and projects in this book, you will need data files that have been created specifically for this book. Your instructor will provide the data files to you. You also can obtain the files electronically from the Course Technology Web site by connecting to *www.cengagebrain.com* and then searching for this book title. Note that you can use a computer in your school lab or your own computer to complete the steps and Hands-On Projects in this book.

Using Your Own Computer

You can use a computer in your school lab or your own computer to complete the chapters, Hands-On Projects, and Case

Projects in this book. To use your own computer, you will need the following:

- **Web browser**, such as Microsoft Internet Explorer 8.0 or later, Mozilla Firefox version 3.0 or later, Google Chrome 7.0 or later, Safari 4.0 or later, or Opera version 10.0 or later.
- **Code-based HTML editor**, such as Adobe Dreamweaver, one of the many shareware editors, or a basic text editor such as Notepad on the PC or SimpleText on the Macintosh.

To The Instructor

To complete all the exercises and chapters in this book, your users must work with a set of user files, called the data files, and download software from Web sites. The data files are included on the Instructor Resources CD-ROM. Students may also obtain them electronically at *www.cengagebrain.com*. Follow the instructions in the Help file to copy the user files to your server or standalone computer. You can view the Help file using a text editor, such as WordPad or Notepad.

After the files are copied, you can distribute the data files for the users yourself, or tell them where to find the files so they can make their own copies of the data files. Make sure the files are set up correctly by having students follow the instructions in the "To the User of the Data Files" section.

Course Technology Data Files

You are granted a license to copy the data files to any computer or computer network used by individuals who have purchased this book.

Acknowledgments

Thanks to the team at Course Technology, Cengage Learning for their hard work and honest desire to produce the best book possible.

Thanks to Lisa Ruffolo, a most talented developmental editor, whose hard work and guiding hand make this a better book.

Thanks to the reviewers who provided plenty of comments and positive direction during the development of this book.

Proposal Package Reviewers:

Robert Burdwel: St. Philip's College

Sean Griffin: North Lake College

Andrew Hunt: Bluegrass Community and Technical College

Julie Mader-Meersman: Northern Kentucky University

Amberly Nowak: University of Saint Francis

Chapter Reviewers:

Ricci Heishman: George Mason University

Alice J. Myatt: Georgia State University

Agatha Taormina: Northern Virginia Community College

Adam Valentiner: College of Southern Nevada

As always, thanks to Diana, Samantha, and my Mom for your patience during the long time it took me to write this edition. Thanks, Sammy, for your design insights and color advice.

Thanks to Sharie Cota and Curly, who let me interrupt their walk in the woods to send me mobile screen captures.

Thanks to Peter Bradley for comments and freeware suggestions.

Thanks to Rob Shain for the use of his Seascapes Aquariums logo.

Thanks to all of you who have emailed me in the past with your comments, critiques, and suggestions. I appreciate and value your ideas.

To those of you who read page proofs, gave feedback on illustrations and content, or provided much-needed encouragement, thank you so much.

This page intentionally left blank

CHAPTER 1

HTML and the Modern Web

In this chapter you will learn about:

- Creating Web pages with HTML
- The history of HTML
- Working with HTML5
- Choosing an HTML editor
- Using good coding practices

In this chapter, you explore how HTML is used along with CSS to create Web pages, and learn the history of how HTML has evolved to its current state. As you will see, designing Web pages has changed dramatically in the last few years. You will examine the latest release of HTML, called HTML5, and see how it can adapt to a variety of Web design needs. You will learn about the new elements and capabilities of HTML5, how to choose the best syntax for the Web pages you are going to create, and how to create correct code. Finally, you consider what type of software tool you can use to create your HTML code, and how to use good coding practices to make sure your work is useful now and in the future.

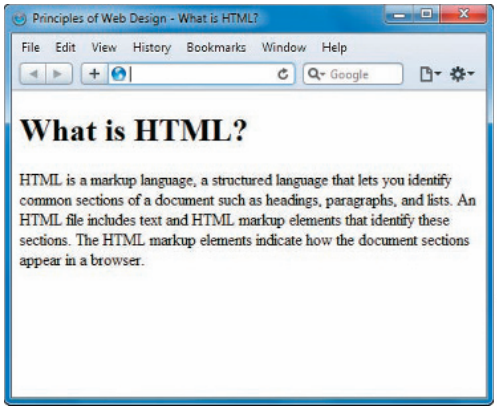
Creating Web Pages with HTML

In today's Web, people shop, trade stocks, watch videos, share photos, play games, communicate via social networks, interact using live chat and video, and much more. They perform all of these activities using **Web pages,** text documents that Web browsers interpret and display. Despite the diversity of content, all of the Web pages on the World Wide Web have one thing in common. They all must be created using some form of the **Hypertext Markup Language (HTML)**. It is astonishing to think that the entire World Wide Web, used every day by millions for shopping, research, banking, and a myriad of other applications is based on a simple text-based markup language that is easy to learn and use.

HTML is a **markup language**, a structured language that lets you identify common sections of a Web page such as headings, paragraphs, and lists with markup tags that define each section. For example, the <h1> element in the following code indicates that the text is a first-level heading:

```
<h1>What is HTML?</h1>
```

Web pages are simply text documents that use HTML to tell the browser how to display each document section. Figure 1-1 shows a basic Web page and the code that is used to create it.




```
<!DOCTYPE html>

<html>
<head>
<title>Principles of Web Design - What is HTML?</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>

<body>
<h1>What is HTML?</h1>
<p>HTML is a markup language, a structured language
that lets you identify common sections of a
document such as headings, paragraphs, and lists.
An HTML file includes text and HTML markup elements
that identify these sections. The HTML markup
elements indicate how the document sections appear
in a browser.</p>
</body>
</html>
```

Figure 1-1 Basic Web page and its HTML code

Structure of a Basic Web Page

An HTML file includes the text that the user sees in the browser, contained within HTML markup elements the user cannot see that identify document sections and elements as shown in Figure 1-2.

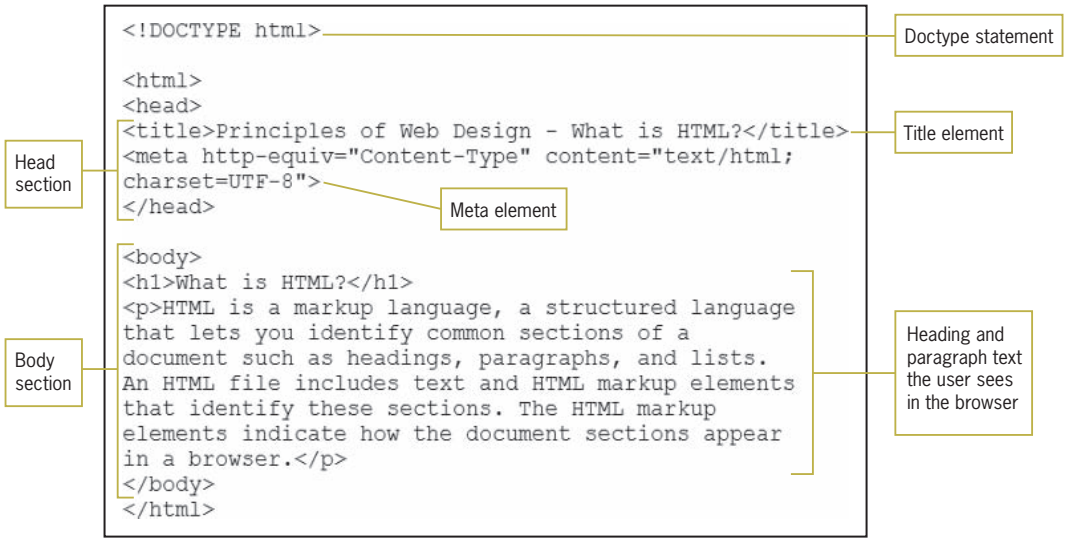


Figure 1-2 Structural elements in an HTML file

The HTML code in this example demonstrates the basic structure of an HTML document. First, the **Document Type**, or doctype for short, specifies the rules for the document language so the browser knows how to interpret the HTML code and display it properly.

After the doctype statement, the opening <html> tag and its closing </html> tag at the end of the page are the root element

of the document. A **root element** is the container element for all other elements in the document.

After the root element is the <head> tag. The two main sections of an HTML document are the head and body sections, represented by the <head> and <body> elements. The head section is the container for all of the descriptive information about the document, including the document title, coding standards, links to external style sheets, and scripting code for interaction. None of the content in the head section appears in the browser window.

The head section contains the important <title> element. This element contains the title of the document, which shows in the title bar of the browser. Document titles should clearly describe the page, contain key terms, and be understandable out of their Web site context. For example, "Home Page" as a title is meaningless outside of the context of its related content. The contents of <title> is a primary source of information for search engines and is often the first text users see in a list of search results. The head section also contains the <meta> element, which defines the content type as type "text/html" and declares the character set for the document.

The body section includes the content that the user sees in the browser window. The body of the document can contain text, images, or audio content, forms for gathering information, interactive content, and hypertext links to other Web resources. Here is where all of the various structural elements that make up HTML come into play. For example, document headings are marked with one of a variety of heading tags, such as <h1> or <h2>, signifying a top-level or secondary-level heading. Paragraph content is marked with the <p> element. HTML offers many elements to expressly mark each section of a document. Appendix A contains a complete list of the HTML elements and their usage.

You should use the HTML elements properly to describe each section of the document based on the logical structure of the document. For example, mark headings as headings, long quotes as <blockquote>, paragraphs as <p>, and so on. In the earlier days of the Web, HTML elements were often misused depending on how they looked in the browser rather than for their structural meaning. Avoid using this type of markup, and express all display information with Cascading Style Sheets, which you will read more about later in this section.

Once you are familiar with the HTML syntax, you will find that one of the best ways to learn new coding techniques is to find a Web page you like and view the source code.

All of the major browsers support a similar action to view the source code. Right-click a blank spot in the browser window and choose a command such as View Page Source or View Source.

HTML in the Browser

The browser interprets the HTML markup elements and displays the results, hiding the actual markup from the user. The user sees only the text "What is HTML?" formatted as a level-one heading and the paragraph text formatted as a paragraph. Figure 1-3 shows the browser's rendition of the HTML code shown in Figure 1-2. Each HTML element contains basic display information to organize and present contents in the browser, such as heading elements that are displayed in a bolder and larger font than paragraph elements. Notice also that the title is displayed in the title bar of the browser.

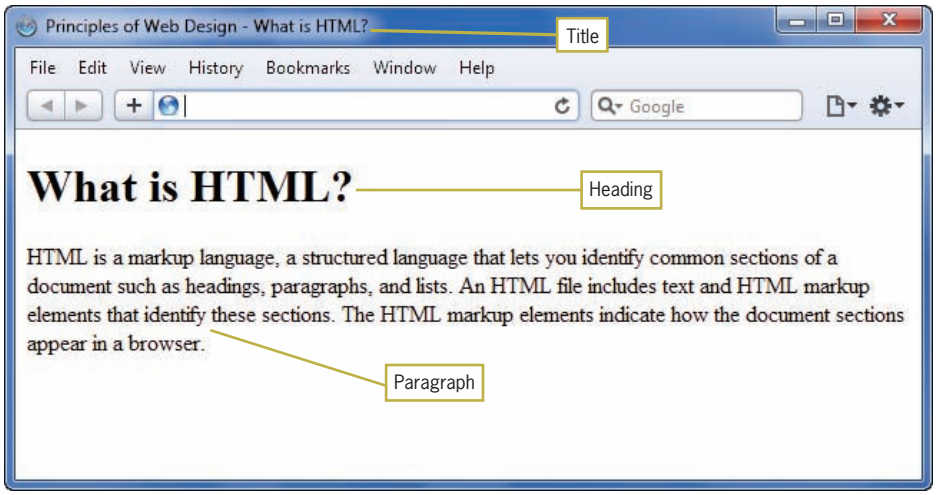


Figure 1-3 Browser interprets the HTML markup and displays the results

As you begin to design and build Web pages, remember that each browser interprets HTML in its own way, based on its **rendering engine**. A browser's rendering engine is software that reads the document's HTML code and associated CSS style information and displays the resulting formatted content in the browser window. Each major browser has its own rendering engine. Although most of your Web pages should look similar in most browsers, it is essential that you test your work in different Web browsers to make sure that your Web pages are rendered as consistently as possible. You will learn more about browser differences in Chapter 2.

Adding Style with CSS

To add presentation information to Web pages, Web designers use a style language called **Cascading Style Sheets (CSS)**. With CSS you can display information for different devices, such as a

cell phone or computer screen, lay out page designs, and control typography, color, and many other presentation characteristics.

Style elements such as <font> were introduced by browser developers to help Web designers bypass the design limitations of HTML. Using elements such as <font> to embed style information within the structure, as is the case in most early Web development, limits the cross-platform compatibility of the content. The display information that is embedded in Web pages is tailored towards one type of display medium, the computer screen. A **style sheet** is a set of style rules that describes the display characteristics of a document. With style sheets, the presentation properties are separate from the content. This accommodates the variety of devices and users that browse the Web, as shown in Figure 1-4.

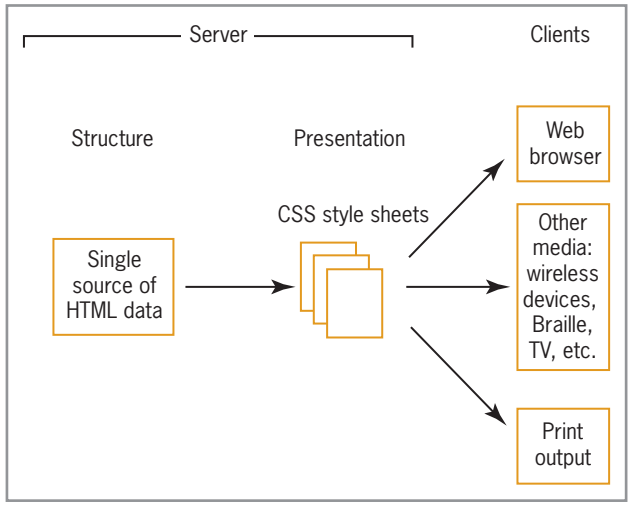


Figure 1-4 Formatting data for multiple destinations

CSS lets you control the presentation characteristics of an entire Web site with a single style sheet document. For example, assume that you want all of your <h1> headings to appear green and centered everywhere on your Web site. Prior to CSS you might have included the following code for every instance of an <h1> element:

```
<font color="green"><h1 align="center">Some Heading Text</
h1></font>
```

Using a CSS rule, you can express the same style as follows:

```
h1 {color: green; text-align: center;}
```

You can place this rule in an external style sheet and then link every page on your site to that style sheet; the single rule controls every <h1> element in your Web site. Later, if you want to change

the <h1> color to red, you simply revise the style sheet rule to change every page on your site.

Through much of Web history, the adoption of CSS as a standard for style has been limited because of poor and uneven support by the major browsers. Modern browsers such as Internet Explorer, Firefox, Opera, Safari, and others offer more complete and consistent support for CSS, freeing designers to work with this powerful style language. Modern Web design requires CSS. You will learn about CSS in later chapters of this book.

Let's revisit the Web page shown in Figures 1-2 and 1-3. Adding some simple CSS code adds style to the page. Notice the style section in Figure 1-5. The style rules specify that the body text for the page will be Arial, heading 1 will have a bottom border, and the paragraph will have a 30-pixel left margin. Figure 1-6 shows the results of the addition of style rules.

```
<!DOCTYPE html>

<html>
<head>
<title>Principles of Web Design - What is HTML?</title>

<style type="text/css">
body {font-family: arial;}
h1 {border-bottom: solid 1px;}
p {margin-left: 30px;}
</style>

</head>

<body>
<h1>What is HTML?</h1>
<p>HTML is a markup language, a structured language
that lets you identify common sections of a document
such as headings, paragraphs, and lists. An HTML
file includes text and HTML markup elements that
identify these sections. The HTML markup elements
indicate how the document sections appear in a
browser./p>
</body>
</html>
```

CSS style section

Figure 1-5 CSS style section contains presentation information

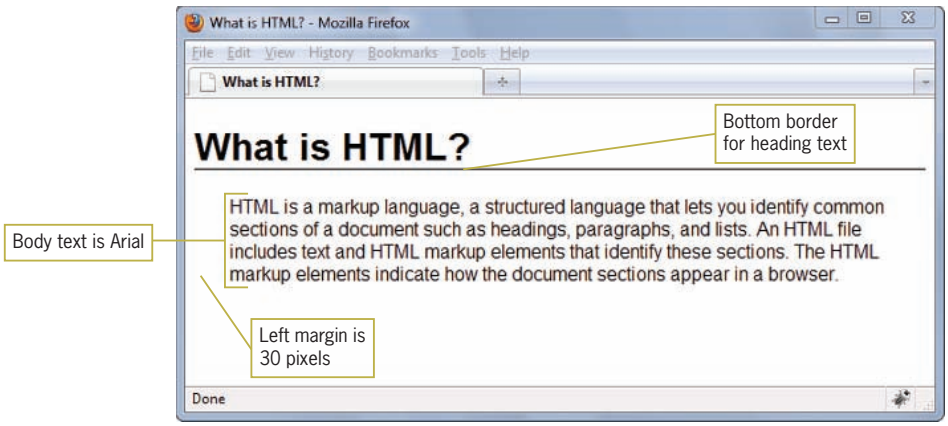


Figure 1-6 Result of adding the CSS style rules

Organizing Information with Hypertext

The most engaging aspect of browsing the World Wide Web is the linking of information on related topics using **hypertext**, a nonlinear way of organizing information. When using a hypertext system, you can jump from one related topic to another, quickly find the information that interests you, and return to your starting point or move onto another topic. As a Web designer, you determine which terms to create as hypertext links and where users end up when they click a link.

On the Web, clickable hyperlinks, which can be text or images, can connect you to another Web page, for example, or allow you to open or download a file, such as a music, image, movie, or executable file. Although the basic one-way nature of a hypertext link has not changed since the Web was developed, the nature of the destination content has changed greatly. The different types of linked content and media continually evolve as the Web continues to grow into an increasingly richer and more interactive environment.

The History of HTML

Now that you've seen the basics of how HTML works, it is important to know how the Web has evolved to its current state. As a Web designer, you will encounter all types of HTML coding practices in the real world. Understanding the evolution of HTML will help you understand various Web design methods. As a Web designer, you may encounter sites that completely comply with current standards, ones that still use design trends and code from years ago, and others that mix coding styles in improbable ways. Many

sites still use table-based designs, which are outmoded and now rendered obsolete by CSS. Web design tools can create all different kinds of code. Proprietary software implementations manipulate code to get the exact result they want. Although it is easy to say that everyone should use Web standards, to be a successful Web designer you need to understand the past, present, and future directions of HTML, coding standards, and common practices, many of which may not match the standards you learn about in this book.

When Tim Berners-Lee first proposed HTML at the European Laboratory for Particle Physics (CERN) in 1989, he was looking for a way to easily manage and share information among scientific colleagues over the Internet. Until this time, the complexity of using the Internet for exchanging messages and sharing files limited its use to groups of specialists in defense, academia, and science.

Berners-Lee joined the ideas of a simple tool for reading the documents (the browser), rules for creating a document markup language (HTML), and a communications protocol that allowed hypertext linking through Uniform Resource Locators (URLs). This accessible, simple interface made using the Internet available to the public. Not only could people read documents, they could easily create them using the easy-to-understand HTML.

As Berners-Lee developed the idea of a Web of documents connected by hypertext links and hosted by computers called hypertext servers, he created a simplified application of the **Standard Generalized Markup Language (SGML)**, a standard system for specifying document structure, which he called the Hypertext Markup Language. HTML significantly reduces the complexity of using SGML to facilitate transmission of documents over the Internet.

When Berners-Lee created HTML, he adopted only the elements of SGML necessary for representing basic office documents such as memos and reports. The first version of HTML included roughly 20 elements that represented basic document structure such as titles, headings, paragraphs, and lists. HTML was intended for simple document structure, not for handling today's varied and complex information needs.

A Need for Standards

After the initial surge of interest in HTML and the Web, a need arose for a standards organization to set recommended practices that would guarantee the open nature of the Web. To meet this need, the **World Wide Web Consortium (W3C)** was founded in 1994

at the Massachusetts Institute of Technology. The W3C sets standards for HTML and provides an open, nonproprietary forum for industry and academic representatives. The various committees that make up the W3C look to expand and set standards for the many new Web technologies that have emerged.

After the W3C was founded, the popularity of the Web grew exponentially. By the mid-1990s, companies started to realize that having a Web presence was vital. Publishing companies started reproducing their paper-based content on the Web. Every advertisement contained a Web address. More and more companies hired print designers to develop Web sites. At this time, Web design was a haphazard affair. HTML was a designer's nightmare, primarily because it was intended for basic page structure, not to create multicolumn print-type layouts. Most computer monitors used a 640 x 480 resolution and many only supported 256 colors, which limited design choices. Multiple browsers, each with their own proprietary elements, competed for market share, and many Web sites were coded primarily for one browser or another. Web designers would manipulate the language in any way they saw fit to achieve a desired design result. After HTML tables were introduced, they became the designer's tool of choice, because they allow the creation of multicolumn layouts. Although designed for data, designers manipulated tables as they needed to achieve the design they wanted.

As the Web grew, designers learned they could get away with manipulating HTML because Web browsers are very forgiving of nonstandard coding. Even if you coded your page incorrectly, you had a good chance that your results would look fine in the browser. If you left out closing tags, or used tags in the wrong order, the page would still be displayed. Web browsers did not output error messages if a page contained a coding error. If your Web site worked with coding errors, you would have no reason to fix them, resulting in an Internet full of Web pages with coding errors.

As development on the Web continued to evolve into more database applications, such as shopping and banking, interoperability became an issue. If every organization codes and manages their Web site content in their own way, exchanging data between organizations becomes difficult. The more everyone followed the same standards, the easier it would be to write browser, application, and database software that "talked" to each other. Jointly developed standards, rather than ones dictated by one vendor, would benefit everyone.

The 1999 release of HTML 4.01 attempted to address a number of these issues. HTML 4.01 supported CSS. This easy-to-use style

language would remove style elements and attributes from the HTML code and replaced them with style rules. This separation of style information from the structure of the HTML document is crucial to the interoperability of HTML, as display information can be customized for the device viewing the Web page, such as a cell phone or computer monitor. HTML 4.01 also deprecated a number of display elements, such as the <font> element. A **deprecated element** means that the element would be removed from future releases of HTML. The W3C was recommending that Web designers stop using these elements in favor of CSS. Many deprecated elements persist to this day, and it is not uncommon to find them in many Web pages and in software programs that create Web pages.

Table 1-1 shows the history of HTML through its releases.

Version	Release Date	Highlights
HTML1.1	1992	First informal draft
HTML 2.0	1995	First release supported by graphical browsers; documents written in HTML 2.0 can still be viewed in all browsers
HTML 3.2	1997	Introduced forms and tables
HTML 4.01	1999	Added support for style sheets, and increased support for scripting and interactivity
HTML5	Future final release, in use today	Latest version adds page layout elements, audio/visual elements, enhanced animation and graphic support

Table 1-1 History of HTML

XML and XHTML: A New Direction

After the release of HTML 4.01, the W3C turned in a different direction for markup languages. In 1997, the W3C released XML, the **Extensible Markup Language**. Although not a direct replacement for HTML, XML has capabilities that are essential to software developers creating applications for the Web. Where HTML is a predefined set of elements that the browser understands, XML lets developers define their own markup language. Software developers could then create elements that matched the data names in their databases. For example, a developer could create a set of elements that described customer information, such

as <name> and <city> that would match the names in a database, easing the transition of data to the Web. Additionally, XML is a stricter language than HTML. XML documents must be syntactically correct to be processed by a software application. This means that only one error in an XML document will keep the document from being processed. This is a very different model from HTML, where the syntax is less strict.

XML code looks very similar to HTML code, with some syntactical differences that you will read about in later in this chapter. The major difference between the two languages is that XML allows you to create elements that describe any type of information you desire. For example, consider that poets might want to create a markup language that expresses the different parts of a poem, as shown in the following code sample:

```
<poem>
<title>An Ode to the Web</title>
<stanza>
<line>So many Web sites</line>
<line>So little time</line>
<line>And all I want to do</line>
<line>Is critique their design!</line>
</stanza>
</poem>
```

Notice that this code looks very much like regular HTML code, except that the tags are not standard, but specific to the type of content they contain. Unlike standard HTML, the browser does not know how to display this information unless CSS style rules are added to specify, for example, that the contents of each <line> element should be displayed in the browser on a separate line or in the color blue.

The W3C saw that XML syntax could provide a solution to the problem of widely varying HTML coding standards, and they started to move in this direction with the evolution of XML-based languages, trying to create a unified syntax under which the entire Web could operate. Towards this end, the W3C reformulated HTML in XML, keeping all the same elements and attributes as HTML 4.01, and named it the **Extensible Hypertext Markup Language**, or XHTML 1.0.

XHTML follows the rules of XML, so it follows the markup rules for XML. In short, XHTML requires that documents follow some basic rules, stated briefly:

- Documents must be well formed.
- All tags must nest properly and not overlap.

- Use all lowercase for element names.
- Always use closing tags.
- Empty elements are marked with a closing slash.
- Attribute values must be contained in quotation marks.

Problems with XHTML

Web designers adopted the new language and syntax, hoping to standardize coding conventions. They adopted CSS to gain benefits in Web site maintenance, interoperability, and adapting content to multiple destination media. The beneficial result is that a majority of commercial Web sites have moved to much leaner, standardized code with all presentation and layout information described by CSS. At the same time, many Web sites and software applications still use legacy style coding conventions. In many instances, relaxed rules had to be applied to projects with legacy content. Also, many Web sites are still created by novices who want to put up a site quickly but who don't want to understand the intricacies of XHTML.

As the W3C continued down the XML-based language path, dissatisfaction increased in the Web developer community. When the W3C issued their first drafts of the XHTML 2.0 recommendation, they announced that XHTML 2.0 would not be backwards compatible with XHTML 1.0 or HTML 4.0. This meant that all of the content on the Web, and all of the work that had been done to develop HTML to its current state would no longer be valid. Further, it dropped familiar elements such as <img> for images and the <a> element for hypertext links, and supported the unforgiving error handling of XML.

A Proposal for HTML5

These decisions moved sharply away from the existing direction of Web development and the long-standing ease of use of HTML. In 2004, an independent group of browser vendors and representatives of the Web development community reacted to the strictness of XHTML2.0 and joined to create a proposal for HTML5. This independent group named themselves the Web Hypertext Application Technology Working Group (WHATWG). After a few years of wrangling over the direction of Web languages, the W3C announced in 2007 that they would restart the effort to

support HTML, and in 2009 they shut down the XHTML 2.0 working group completely.

The W3C's response to the development community's desires for the next generation of HTML highlights the unusual nature of the Web, where standards are not dictated by one vendor, but rather decided upon and influenced by the people who work with HTML and Web design day to day. As you will see later in this chapter, HTML5 supports standards-based coding, is compatible with both XHTML 1.0 and HTML 4.01, and supports new elements for better structuring of Web pages. HTML5 is adaptable to the modern interactive Web, where Web pages are not just static documents, but applications.

Working with HTML5

In this section, you learn about basic HTML5 markup and the two different syntaxes allowed within HTML5. You will learn how to choose the correct syntax for your Web pages, and review the new elements in HTML5. You will see how the new HTML5 layout elements will work, and how HTML5 supports interaction in the browser. However, keep in mind that if you are coding Web pages, you will have to wait for browser support before you start adding new HTML5 page layout elements to your code.

HTML5 is the fifth major revision of HTML. It comes long after the last major revision, which was in 1999. HTML5 attempts to address the shortcomings of previous versions of HTML, while addressing the needs of modern Web design and the application-based future of the Web. HTML5 is intended to not only describe page structure and layout, but offers a variety of new features:

- Logical layout elements, such as <nav> (for a navigation bar), <section>, <header>, and <footer>
- Elements designed for different types of rich media, including <video> and <audio>
- Animations that play directly in the browser window without plug-ins using the new <canvas> element
- Support for applications in the browser, including drag and drop, local data storage, and background data processing

HTML5 also removes some features from previous versions of HTML, as described in the following list:

- All display elements have been removed in favor of CSS for presentation.
- Framesets and frames have been removed because of their impact on accessibility.

HTML5 is compatible with both HTML 4.01 and XHTML 1.0 and supports both the "looser" coding style associated with earlier HTML versions and a stricter syntax that is based on XML.

HTML5 looks almost exactly like previous versions of HTML. Figure 1-7 shows a sample of HTML5 code. Note two important conventions in this sample:

The HTML5 <!DOCTYPE> statement is less complicated than in previous versions of HTML.

The <meta> element specifies the document content type and character set. Many pages leave out this critical piece of information that tells the browser how to correctly interpret your HTML code.

```
<!DOCTYPE html>

<html>
<head>
<title>Principles of Web Design - What is HTML?</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>

<body>
<h1>What is HTML?</h1>
<p>HTML is a markup language, a structured language
that lets you identify common sections of a
document such as headings, paragraphs, and lists.
An HTML file includes text and HTML markup elements
that identify these sections. The HTML markup
elements indicate how the document sections appear
in a browser.</p>
</body>
</html>
```

HTML5 doctype statement

<meta> element specifies content type and character set

Figure 1-7 Sample HTML5 document

HTML5 Loose and Strict Syntaxes

HTML5 offers two syntaxes. One is based on HTML syntax, and the other on stricter XML syntax rules, which makes it compatible with XHTML.

HTML Version of HTML5

The HTML version of HTML5 is more relaxed and allows authors to use shortcuts in their code. Figure 1-8 shows the looser syntax with two examples of code shortcuts. Notice that the content type specifies HTML.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Loose Syntax</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>
<h1>Example</h1>
<p class=content>HTML5 allows a looser syntax.
</body>
</html>
```

MIME type specifies HTML

<p> element has no closing tag

Attribute value is not quoted

Figure 1-8 Looser HTML5 syntax

Notice in this code that the <p> element has no closing tag, and the class attribute has no quotes around the content value. This more relaxed version of HTML5 is backwards compatible with HTML 4.01. The MIME Type, described later in this section, declares the document as an HTML document.

XHTML Version of HTML5

The stricter syntax rules of HTML5 are consistent with XHTML syntax, as shown in Figure 1-9. XML syntax rules are applied to the code and the two shortcuts removed. The XHTML Namespace qualifier and XHTML MIME type declare this as an XHTML document.

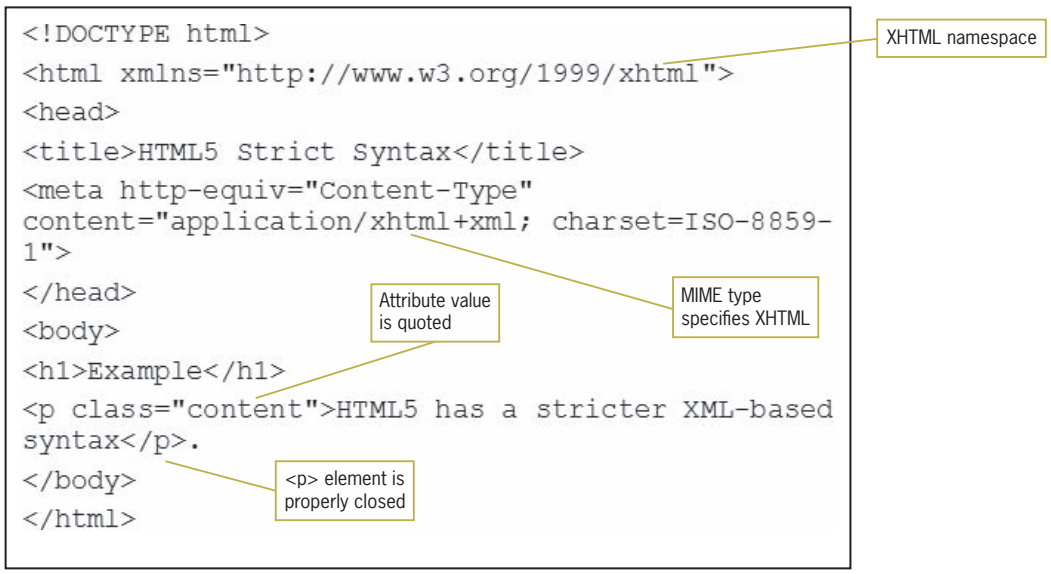


Figure 1-9 Stricter HTML5 syntax

Choosing the Correct Syntax

HTML5 allows a mixture of these two types of syntax into one document, called a polyglot document by the W3C. Polyglot means "mixed languages," and the name is appropriate for the type of HTML syntax that is the most usable, that is, declaring the document as HTML, but using the stricter syntax rules of XML. This lets you standardize and consistently code to a stricter, more consistent coding convention for any type of professional Web site.

The looser HTML syntax that is allowed in HTML5 is appropriate for anyone building a noncommercial Web site. However, in a professional Web development environment, it is a best practice to code using syntax that follows the basic XML syntax rules, which you will read more about later in this chapter. With XML syntax, the code you create for Web content can have multiple purposes and potentially be used in a variety of display and application environments. You can choose to code to XML standards but still declare the document MIME type as HTML, so that error handling is not a problem.

For example, it is now commonplace in the publishing industry to repurpose content that was originally destined for Web publication. This content can be organized and used in a content

management system that allows output to be printed or displayed in different environments that meets users' individual needs. The multipurposing of content is often called **single-sourcing**, where one source of content is maintained but disseminated to different users or devices. Data in a single location can be maintained and updated more easily. The data also has greater value because it can be displayed and used in different ways.

Another reason for choosing XML syntax in your HTML5 code is to ensure consistent coding practices across an organization. Web site development should always formulate and apply consistent coding conventions across all Web content to ensure uniformity. With XML syntax, the rules are already created and just need to be followed. XML rules can be easily tested for compliance by using a **validator**, software that checks an HTML document for syntax errors. You can use either an online validator or a built-in validator that is common to most HTML development tools. If you create coding conventions and stick to them, your Web content will be more adaptable to different purposes, compatible with more software applications, and better prepared for future use.

Choosing the Correct Document Type and MIME type

The key to displaying your Web pages correctly in the browser is stating the correct <!DOCTYPE> and MIME type.

The DOCTYPE Statement

The <!DOCTYPE> statement originates in SGML, and was used to point to a **Document Type Definition (DTD)**, which contains the elements, attributes, and syntax rules for a language like HTML or XML.

HTML5 is no longer based on SGML, so the <!DOCTYPE> statement primarily exists to make sure that browsers display the page in standards mode. When you include a document type, the browser operates in "standards mode," presenting the document using W3C rules. If there is no document type, the browser operates in "quirks mode," which renders the page like an older browser, allowing the "quirky" code from the earlier days of Web development.

The standard doctype statement for HTML5 looks like this:

```
<!DOCTYPE html>
```

MIME Type

The **Multipurpose Internet Mail Extensions (MIME)** originated as a standard for e-mail, but has grown to defining content types for the Web. It is the MIME type that determines the type of document you are presenting. Even in a document written in strict XHTML, if the MIME type is set to text/html, then the document is served as HTML.

Every document should contain a <meta> element in the <head> section that specifies the content type and character set. The content value is the document's MIME type. The document's character set is specified as charset=utf-8, which is the standard character set for the Web.

Your <meta> element should look like this:

```
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
```

Putting all of this together, the beginning of all HTML5 documents should look like the following code, with the addition of a page name in the <title> element:

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
```

You may want to create a blank template file that has this code in it, and then copy it and rename it every time you need to create a new HTML file.

Creating Syntactically Correct Code

You should follow six basic rules to create syntactically correct HTML5 code. These are the same rules used for both XML and XHTML, applied in an HTML environment, and were listed earlier in the introduction to XHTML.

- Documents must be well formed.
- All tags must nest properly and not overlap.
- Use all lowercase for element names.
- Always use closing tags.
- Empty elements are marked with a closing slash.
- Attribute values must be contained in quotation marks.

Documents Must Be Well Formed

A **well-formed document** is one that adheres to the syntax rules described in this section.

All Tags Must Nest Properly and Not Overlap

You can nest HTML elements, but they must not overlap. Each set of opening and closing tags must completely contain any elements that are nested within the set. For example, the following is incorrect syntax:

```
<p><strong>some text... </p></strong>
```

The closing tag for the bold attribute must come before the closing tag for the paragraph element. The correct nesting syntax follows:

```
<p><strong>some text...</strong></p>
```

Use All Lowercase for Element Names

Even though HTML5 allows uppercase and lowercase element names, it is considered good coding practice to always use lowercase. Use all lowercase characters for element and attribute names when writing your code, this will ensure that your code can be XHTML compatible if it needs to be used in some type of content management or application processing environment.

Always Use Closing Tags

In the looser HTML5 syntax, certain elements such as the <p> element are allowed optional closing tags. For example, the following two <p> elements do not have closing tags:

```
<p>This is the first paragraph.
<p>This is the second paragraph.
```

Even though HTML5 allows this, a much better practice is to always close all elements. The following example shows this syntax.

```
<p>This is the first paragraph.</p>
<p>This is the second paragraph.</p>
```

Empty Elements Are Marked with a Closing Slash

Empty elements must be marked "empty" by a slash (/) in the single tag. For example, the <br> element, becomes <br />. The <img> element looks like the following:

```
<img src="photo.jpg" />
```

Notice the closing slash. Older browsers ignore this, so you can convert empty elements to be XHTML-compliant without worrying if your pages are displayed properly.

In previous versions of HTML and XHTML, authors left a blank space in front of the slash, as in <br />. This is no longer necessary in HTML5.

Attribute Values Must Be Contained in Quotes

In the looser HTML5 syntax, attribute values do not have to be quoted as shown in the following example:

```
<p class=copy>HTML5 is the newest Web language.</p>
```

Even though HTML5 allows this, your code is more compatible when you always contain all attribute values within quotes. The preferred syntax follows:

```
<p class="copy">HTML5 is the newest Web language.</p>
```

Element Categories

In HTML5, elements are divided into categories by use, as shown in the following list. Some elements may fall into more than one category.

- Metadata content
- Flow content
- Sectioning root
- Sectioning content
- Heading content
- Phrasing content
- Embedded content
- Interactive content
- Transparent

Metadata Content

These are the elements that reside in the head section of the document, such as <title>, <script>, and <style>. These elements contain the document **metadata**, which is information about the document itself, such as how to present the document, or what other documents are related to this one, such as style sheets.

Flow Content

Flow content elements are most of the elements that are used within the body section of the document. This category includes all of the standard HTML tags, such as <p>, , and <blockquote>, as well as the newer page layout elements introduced in HTML5 such as <article>, <header>, and <footer>.

Sectioning Root

Content on Web pages is divided into sections. The <body> element is the root or parent element of all content sections, making it a sectioning root. Other elements in this category include <blockquote> and <td>.

Sectioning Content

Sectioning content divides a document into sections, each of which can have its own set of headings. These elements group sections of content on the page. Elements in this category include <section>, <article>, and <nav>. Sectioning elements are an exciting feature of HTML5, and most of these elements are new. Make sure that browsers support these new elements before beginning to use them in your Web pages. In the meantime, continue to use the element with classes to create sections. You will read more about this in Chapter 7, Page Layouts.

Heading Content

This category includes the heading elements, <h1> thru <h6>, plus the new <header> element for creating heading sections.

Phrasing Content

Phrasing content includes elements that are used within lines of text in a document. These include , , and . These elements were called inline elements in HTML 4.01.

Embedded Content

Embedded content elements load external content into the Web page. This content can be images, videos or audio files, or Adobe Flash files. The <img> element as well as the new <audio> and <video> elements are part of this category.

Interactive Content

Interactive elements let users interact with the Web page content. Interactivity depends on the user's browser and input device, such as a mouse, keyboard, touch screen, or voice input. Elements in this category include the <a> element, which creates clickable hyperlinks, plus the <audio> and <video> elements if the user can control the content, such as by using play or pause buttons. Flash animations and other content types, such as presentations or Web-based learning, also accept user interaction. Form controls are also part of this category.

Transparent

Some elements have transparent content models, which means that their allowed content is inherited from their parent element. They may contain any content that their parent element may contain, in addition to any other allowances or exceptions described for the element. For example, the <a> element often occurs within a <p> element parent, and will usually follow the content rules of its parent element.

New Elements in HTML5

HTML5 has a number of new elements, as listed in Table 1-2. Not all of these elements are supported by current browsers. Make sure to test carefully and check for browser support before using these new elements.

Element	Description
<article>	A section for the main page content, such as a newspaper or magazine article, blog entry, or any other independent item of content
<aside>	A section for side content such as a pull quote or other content related to the main article
<audio>	Contains sound, streaming audio, or other aural content
<canvas>	Lets scripting applications dynamically render graphics, animations, or other visual images
<command>	Defines a command action for interaction with the user
<datalist>	Contains drop-down list content for older browsers; used for legacy compatibility

Table 1-2 New Elements in HTML5 *(continues)*

(continued)

<details>	Contains additional information or controls that the user can obtain on demand
<embed>	Represents an insertion point for an external application or interactive content
<figure>	Contains an image or graphic with an optional caption using the <legend> element
<footer>	Typically contains information such as who wrote the Web page, links to related documents, and copyright notices
<header>	Typically contains headings and subheadings that describe the page content
<mark>	Includes a string of text in a document marked or highlighted for reference purposes
<meter>	Includes a measurement within a known range, or a fractional value
<nav>	Contains primary navigation links to other pages or to content within the page
<output>	Contains the result of a calculation
<progress>	Represents the completion progress of a task
<rp>	Stands for ruby parentheses, which hide ruby text <rt> from browsers that do not support the <ruby> element
<rt>	Contains ruby text, used with the <ruby> element
<ruby>	Allows markup of text content with ruby annotations. Ruby annotations are descriptive strings of text presented next to base text, primarily used in East Asian typography as a guide for pronunciation or to include other annotations.
<section>	Specifies a generic section of a Web page; a section, in this context, is a thematic grouping of content, typically with its own heading
<source>	Specifies multiple media resources for media elements
<time>	Contains a date or time
<video>	Contains video content

Table 1-2 New Elements in HTML5

Attributes in HTML5

Elements can contain attributes that set properties for an element. In the following code sample, the element has a class attribute with a value of *article*.

```
<div class="article">
```

In earlier versions of HTML, a wide variety of attributes contained presentation information, such as the size, color, or alignment of text. In HTML5, all display information is specified with CSS, so far fewer attributes are necessary. Some elements have specific attributes, while other attributes are global and can be applied to any element. Some of the more important global attributes include style, title, and class. You will find a complete list of HTML5 attributes in Appendix A.

Obsolete Elements in HTML5

Many elements have been removed from HTML5, most of which were used for presentation effects that are better handled with a CSS style. Examples of elements that are obsolete in HTML5 include some that are commonly avoided in Web design, such as <font>. Also, the <frameset> and <frames> elements are no longer available, so frame-based sites are obsolete. The <iframe> element is still available to embed a page within a page. You will find a complete list of obsolete elements in Appendix A.

Using HTML5 Elements for Page Structure

Generally, all Web pages share some common characteristics. For example, most Web pages have some type of header, navigation, article and figure content, footers, and possibly sidebars that contain related content. In current Web design, and as you will learn in later chapters, page layout sections are currently marked up using a combination of elements and id or class names, as shown in Figure 1-10.

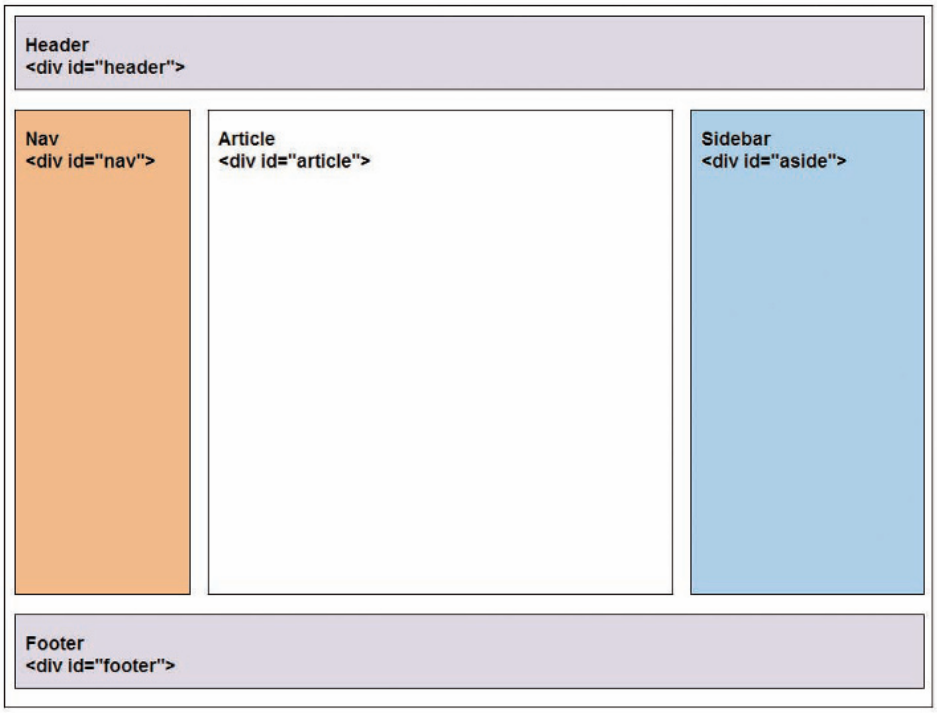


Figure 1-10 Five page sections using elements and id names

The id names shown in Figure 1-10 may vary, but generally all of the page components shown occur in one form or another on every Web site. HTML5 standardizes the naming conventions for each of these sections and provides an element for each and other document components as well.

HTML5 offers a new set of elements for describing document structure. These new elements are based on the methods that have become popular on the Web as browsers offered increasing support for layouts created with CSS. These techniques supplant the use of tables for page layout, instead using the division element, or , to structure the page. The elements are named with an id or class attribute, such as:

```
<div id="article">This is the main content of the Web page</div>
```

HTML5 replaces the use of with named elements to structure the page. The <article> element can be used instead of the element, for example:

```
<article>This is the main content of the Web page</article>
```

Using HTML5, the Web page shown in Figure 1-10 can now be marked up as shown in Figure 1-11.

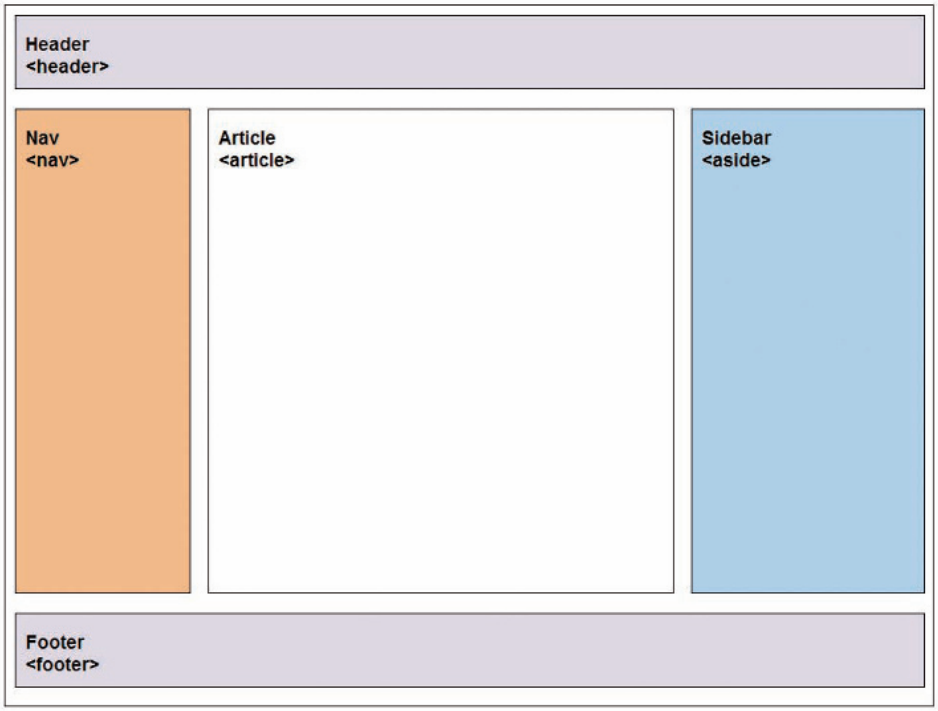


Figure 1-11 Page sections using HTML5 elements

The HTML5 elements that are designed for page layout include the following:

- <header>—Contains the page header content
- <nav>—Contains the navigation elements for the page
- <article>—Contains the primary page content
- <section>—Defines sections or groupings of the page content
- <aside>—Contains additional content such as a quote or sidebar
- <figure>—Contains images for the article content
- <footer>—Contains the page footer content, typically information such as copyrights, policies and legal information, and contact information.

These elements offer benefits for building page layouts. They standardize the naming conventions for parts of a Web page, allowing

software tools to apply consistent rules for content management, presentation, and processing. They allow Web pages to share information across multiple delivery platforms and presentation devices.

The <section> element is for thematic groupings of content, each with its own heading structure. This means that each section can have up to six headings (<h1> through <h6>), which can create more complex page structures. The <section> element also allows automatically generated tables of content or outlines of page content.

As with all new advances in HTML, the browsers need to catch up to support the latest enhancements. This means developers have to add new code and prepare new releases of their software, and these new browser releases have to be accepted and downloaded by the general Web-browsing public. Remember that if you are coding Web pages, you will have to wait for browser support before you start adding these new page layout elements to your code. In the meantime, the naming conventions developed for these new elements, such as <nav> or <article>, can be used now as class names until browser support catches up. You will learn more about this in Chapter 7.

Interactive Capabilities in HTML5

HTML5 supports a number of new features for applications and interaction with users. Browser support varies for these new features, so evaluate and test them carefully before implementation.

Audio and Video

The <audio> and <video> media elements let developers embed audio or video streams into a Web page.

Drawing Canvas

The <canvas> element provides a bitmap drawing area for displaying graphs, drawings, games, or other visual elements dynamically in the browser.

Background Application Processing

"Web workers" are background scripting processes that can run in the browser while the user is performing other actions. This means that calculations, data transfers, and other background processing are available to support more complex applications.

Local Data Storage

As browser-based applications become more complex, they need to be able to store data locally on the user's machine, rather than having to interact with the server each time data needs to be read or written. This task is currently handled by **cookies**, small pieces of text-based data that is stored on the user's machine. Cookies transfer only small pieces of information with every request to the Web server. Part of the HTML5 effort is a recommendation called Web Storage that specifies how larger amounts of data can be stored and retrieved from the user's computer by Web applications.

Choosing an HTML Editor

You can create or generate HTML code to build Web pages in many ways. The two ways to create HTML are using a code-based editor or a WYSIWYG (What You See Is What You Get) editor. Some HTML editors combine a WYSIWYG view and a code-based view. Others have built-in validators that check your code for syntax errors. Many have FTP (file transfer protocol) clients that publish your Web pages to your Web host. You will read more about publishing your Web site in Chapter 3.

You do not have to spend a lot of money to acquire an HTML editor. Many are low-priced shareware and others are available free. You can also use simple text editors as HTML-editing programs. Notepad, the basic text editor that comes with Windows versions from 95 to Windows 7, is still available for a quick edit, to view page code, or to create entire Web pages. On the Macintosh, you can use TextEdit to create HTML files. Many sites on the Web are coded using these text-editing tools, which are easy to use and still relied upon by many HTML authors. They also are the best way to learn HTML because you have to enter every tag by hand. However, fewer designers use simple text editors now that increasingly robust HTML-authoring packages have appeared.

Full-featured HTML-editing programs include Adobe Dreamweaver and Microsoft Expression Web (which replaces FrontPage), to name two. Some code-based HTML editors, such as Adobe HomeSite, forgo a WYSIWYG approach. They have become popular because they include many powerful enhancements that Notepad lacks, such as multiple search-and-replace features and syntax checking, while still allowing you to manipulate code at the tag level. The most recent authoring tools offer syntax validation and code conversion as well, which can greatly

lessen the tasks of cleaning up older code to match the newer HTML syntax rules. See Chapter 3 for a list of HTML editors.

Many of the latest office applications also convert documents to HTML. For example, you can create a flyer in your word processor and export it to create an HTML page. You can even create slides in Microsoft PowerPoint and export them to HTML. This hands-off approach leaves much to be desired for an HTML author, however, because you give up control over the finished product. In addition, the practice of converting content from a program such as Microsoft Word to HTML is notorious for creating substandard HTML code.

As with browsers, authoring packages interpret tags based on their own built-in logic. Therefore, a page that you create in an editing interface may look quite different when displayed in a browser. Furthermore, many editing packages create complex, substandard code to achieve an effect specified by the user. This complex code can cause compatibility problems across different browsers. HTML authors who are accustomed to coding by hand (in Notepad or another text editor) often are surprised to see the complex code an HTML editing package generates.

To code effectively with HTML, you must be comfortable working directly at the code level. Though you may choose to use one of the many editing packages to generate the basic layout or structure for your page or to build a complex table, be prepared to edit the code at the tag level to fix any discrepancies. You probably will end up working with a combination of tools to create your finished pages.

Using Good Coding Practices

In the past, Web designers wrote nonstandard code, employing a "whatever works" mentality to trick the browser into presenting the results they wanted. Some examples of this are using the heading tags <h1> through <h6> solely for their font sizes rather than as logical headings, or manipulating the <table> elements into a layout tool instead of a container for data as they were intended.

These are bad habits that are best left behind. This section provides the following guidelines for creating code that ensures the greatest standards-compliance, presentation, and usefulness of your content:

- Stick to the standards.
- Use semantic markup.
- Validate your code.

Stick to the Standards

The best way to create well-coded Web sites is to strictly follow the standards set by the W3C. This approach provides the greatest acceptance and most uniform display of your content across multiple browsers and operating systems. This "best practices" method of coding is widely supported among sites that are interested in the widest accessibility. Following the W3C standards does not mean that your site has to be visually uninteresting, although you may have to sacrifice the latest multimedia enhancements. Reliable visual and information design techniques, along with the use of CSS, can let you overcome many functional limitations.

Coding with standards and respecting the W3C mandate to separate content structure from presentation information makes your content more accessible and portable to different devices and destinations. Who would have envisioned that, at the outset of the Web, content would be delivered on cell phones or televisions? Who can imagine what devices will be used in the future? Content that is designed to standards is more meaningful to search engines, has better accessibility, is displayed more consistently in multiple browsers, and has a longer life and greater chance of being accessible in future applications.

The Web Standards project (*www.webstandards.org*) is "a grassroots coalition fighting for standards that ensure simple, affordable access to Web technologies for all." This site is a good place to get started on reading more about the history, trends, and current state of Web standards development.

Use Semantic Markup

Semantic markup is descriptive markup that identifies the intended use of document sections. Semantic markup accurately describes each piece of content. Although this may sound like an obvious use of markup elements, until recently this logical use of the HTML elements was largely ignored by the Web development community. For example, HTML authors know that an <h1> element signifies a block of text as a first-level heading, but many times this element is used simply for its ability to increase the font size of text. The <ul> element, which indicates a bulleted list, was used for its indenting characteristics. When you use semantic markup, the document elements match the meaning and usage of the document sections; a <p> signifies a paragraph, a <blockquote> is for a lengthy quotation, and so on. Semantically correct markup is readable not only by humans but by software as well, lending itself to improved recognition by search engines, news agents, and accessibility devices.

Semantic markup is the basis for the evolution of the Web into a universal medium for data, information, and knowledge exchange as described by the W3C in their Semantic Web Activity group (*www.w3.org/2001/sw*). Although this vision of the Web is still in early development, the emphasis on correct usage of markup elements benefits the accessibility and longevity of your Web content.

Validate Your Code

Another step towards standards compliance is to validate your code. **Valid code** conforms to the usage rules of the W3C. For example, a basic HTML rule states that only certain elements can appear in the head section. The lack of valid code is a major problem for the future of a standards-based Web. A recent survey of 2.4 million Web pages found that 99 percent did not meet W3C standards. Although this number may have decreased since the study, a quick survey of almost any Web site shows that few have valid code.

Valid code enhances browser compatibility, accessibility, and exchange of data. Whether you write HTML or XHTML code, you should validate your code to make sure it conforms to W3C standards. Validation is so easy to perform, it is hard to understand why many Web designers apparently ignore it. To validate your code, simply use a software program called a validator to read your code and compare it to the rules in the DTD. The validator generates a list of validation errors. Many HTML editors contain built-in validators, or you can use a Web-based validator such as the W3C validation service at *http://validator.w3.org*, which can validate HTML5.

Using a validator is an eye-opening experience for any Web designer. The most common mistakes that make your code invalid include:

- No doctype declaration
- Missing closing tags
- Missing alt attributes in <img> elements
- Incorrect tag nesting
- Unquoted attributes

If you compare this list to the XML syntax rules listed earlier in this chapter, you can see how moving to standardized coding practices helps clean up most of these common coding errors.

A new offering from theW3C is Unicorn, an all-in-one validator that checks both HTML and CSS code for errors on any Web page you specify. You can also upload a file if you do not have a live Web site (*http://validator.w3.org/unicorn*).

Migrating from Legacy HTML to HTML5

If you are moving your existing Web site to HTML5, the transition should be a gradual process rather than an abrupt shift to the new language. The looser syntax of HTML lets you start to adopt the newer syntax while keeping legacy HTML code such as attributes that control page and link colors. As you migrate closer to

HTML5, you will be cleaning up code on existing pages, planning coding conventions for new pages, removing deprecated elements, and moving display information to CSS. Eventually you will be creating Web pages that contain only structural information, with all display information kept separately in CSS files.

The following list outlines steps you need to take to migrate from legacy HTML to HTML5:

1. *Evaluate existing code*—Check for basic compliance with consistent syntax rules. Are closing tags included? Are all tags lowercase? Are attributes quoted? How much cleanup work is necessary to make the code well formed? Most of this work can be automated in the various HTML editing programs.
2. *Evaluate existing presentation information*—How much of your code includes obsolete elements such as <font> and obsolete attributes such as "bgcolor," "face," and "size"? On many sites, this information can make up as much as 50 percent of the existing code. Start thinking about how you can convert these presentation characteristics to CSS.
3. *Create coding conventions*—Create coding conventions and follow them throughout the site. Make sure that new content added to the site follows the new coding and CSS standards. The more you standardize, the easier your maintenance chores become.
4. *Start using CSS*—Start by building simple style sheets that express basic characteristics such as page colors, font family, and font size. Consider using more advanced CSS options such as classes that allow you to name and standardize the various styles for your site. As you build style rules, start to remove the existing display information in the site.
5. *Test for backward compatibility*—Remember to test in older browsers to make sure that your content is legible and readable. Test carefully with your CSS style rules to make sure that they are supported in older browsers.

Chapter Summary

Many variables affect the way users view your Web pages. As an HTML author, your goal should be to code pages that are accessible to the largest audience possible. As you plan your Web site, make the following decisions before implementing your site:

- Make sure to check for support of new HTML5 elements, and test carefully before adding them to your Web site.
- Use the HTML5 naming conventions to name the content sections of your site, even if only using them as class or id names. This will help you ease the migration to the new HTML5 page layout elements as they become supported by browsers.
- Use Cascading Style Sheets. The style enhancements and control offered by this style language are impressive, but are not evenly supported by older browsers. Implement CSS gradually, testing for browser compatibility as you go.
- Decide whether to code to the XML standard. If you are starting a new Web site, your best choice is to code to this new standard. If you are working with an existing Web site, decide on the most expedient method for upgrading your existing code to XML standards to ensure future compatibility with new tools and browsers.
- Use good coding practices by writing standards-based markup that always includes a document type and content type and is correctly validated.
- Choose the type of editing tool needed to create your HTML code. If you choose a WYSIWYG editor, make sure it creates standards-based code. Avoid using word-processing programs to create Web pages.

Key Terms

Cascading Style Sheets (CSS)—A style language, created by the W3C, that allows complete specifications of style for HTML documents. CSS allows HTML authors to write style rules that affect the display of Web pages. CSS style information is contained either within an HTML document, or in external documents called style sheets.

cookie—A small piece of text-based data that a Web page stores on the user's machine. Cookies transfer small pieces of information with every request to the Web server.

deprecated element—An element that the W3C has identified as obsolete.

Document Type (doctype)—The section of an HTML document that specifies the rules for the document language so the browser knows how to interpret the HTML code and display it properly.

Document Type Definition (DTD)—A set of rules that contains all the elements, attributes, and usage rules for the markup language you are using.

Extensible Hypertext Markup Language (XHTML)—XHTML is HTML 4.01 reformulated as an application of XML.

hypertext—A nonlinear way of organizing information. When you are using a hypertext system, you can skip from one related topic to another, find the information that interests you, and then return to your starting point or move on to another related topic of interest.

Hypertext Markup Language (HTML)—The markup language that defines the structure and display properties of a Web page. The HTML code is interpreted by the browser to create the displayed results. HTML is an application of SGML. *See also* Standard Generalized Markup Language.

markup language—A structured language that lets you identify common elements of a document such as headings, paragraphs, and lists.

metadata—Information about the document itself, such as how to present the document, or what other documents are related to the current one, such as style sheets.

Multipurpose Internet Mail Extensions (MIME)—Originally a standard for e-mail, MIME now defines content types for the Web. It determines the type of document presented in an HTML file.

rendering engine—A program contained in every browser that interprets the markup tags in an HTML file and displays the results in the browser.

root element—The container element for all other elements in the document. In an HTML document, the root element is <html>.

semantic markup—Descriptive markup that identifies the intended use of document sections. Semantic markup accurately describes each piece of content.

single-source—To create content that can serve multiple purposes and be distributed to different users or devices.

Standard Generalized Markup Language (SGML)—A standard system for specifying document structure using markup tags.

style sheet—A set of style rules that describes a document's display characteristics. There are two types of style sheets: internal and external.

valid code—Markup code that conforms to the usage rules of the W3C.

validator—A software program that checks an HTML document for syntactical errors.

Web page—A text document that is interpreted and displayed by Web browser software.

well-formed document—A syntactically correct XML or XHTML file.

World Wide Web Consortium (W3C)—Founded in 1994 at the Massachusetts Institute of Technology to standardize Web markup languages. The W3C, led by Tim Berners-Lee, sets standards for markup languages and provides an open, nonproprietary forum for industry and academic representatives to add to the evolution of HTML.

Review Questions

1. What does the <!DOCTYPE> statement specify?
2. What is the function of the root element?
3. What are the main sections of an HTML document?
4. Why is the content of the <title> element important?
5. How should display information be expressed for a Web page?
6. What function does the browser's rendering engine perform?
7. What are the advantages of using an external style sheet?
8. What feature distinguishes XML from HTML?
9. What are the two HTML5 syntaxes?
10. What is a well-formed document?
11. What is document metadata?

12. Where is metadata stored?

13. List three new HTML5 elements that are designed for page layout.

14. What is the function of the new HTML5 <canvas> element?

15. What does *semantic markup* mean?

Hands-On Projects

1. Download and install the latest versions of the following browsers onto your computer as necessary:
   - Internet Explorer
   - Firefox
   - Opera
   - Safari
   - Google Chrome

2. Build a basic HTML file and test it in the browser.
   a. Copy the **project1.htm** file from the Chapter01 folder provided with your Data Files to the Chapter01 folder in your work folder. (Create the Chapter01 folder, if necessary.)
   b. In your HTML editor, open **project1.htm** and examine the code. Notice that only the basic HTML elements are included to create the head and body sections of the document.

```
<!DOCTYPE html>

<html>
<head>

</head>
<body>



</body>
</html>
```

c. Add the <title> element with a title for your document and a <meta> element that defines the document type as shown in color in the following code.

```
<!DOCTYPE html>

<html>
<head>
<title>Web Page Project 1</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>



</body>
</html>
```

d. Add an <h1> element with a heading for the document. The <h1> element must be contained within the <body> element.

```
<!DOCTYPE html>

<html>
<head>
<title>Web Page Project 1</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>
<h1>The World Wide Web</h1>



</body>
</html>
```

e. Save your file and view it in your browser. It should look like Figure 1-12.

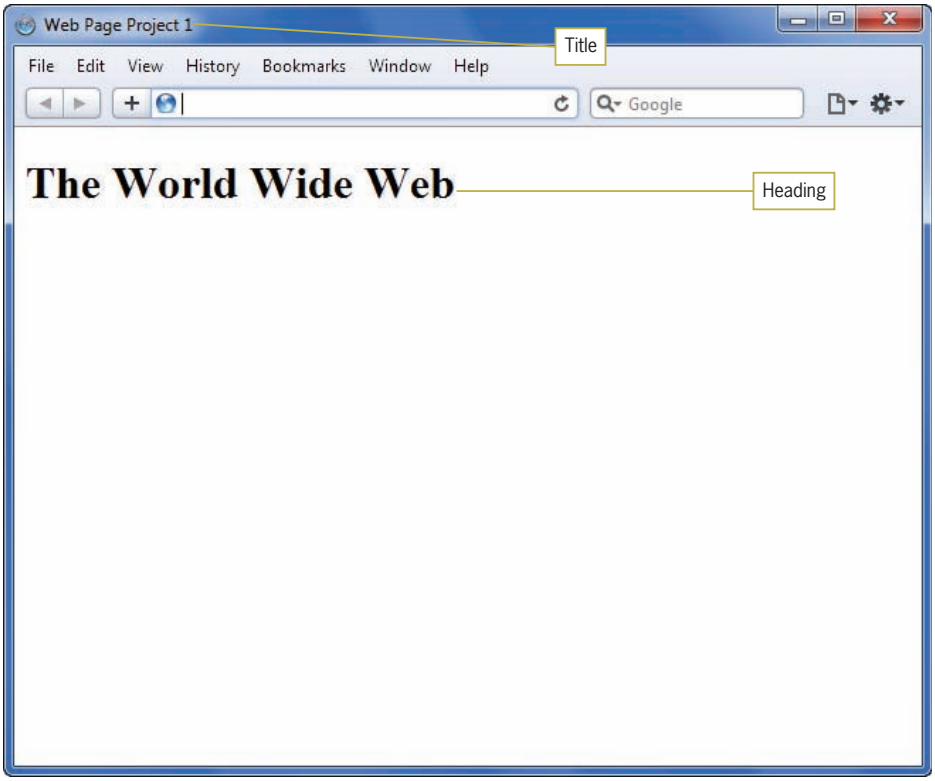


Figure 1-12 Adding a title and heading to the Web page

f. Add two paragraphs of content immediately following the <h1> element as shown.

```
<!DOCTYPE html>

<html>
<head>
<title>Web Page Project 1</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>
<h1>The World Wide Web</h1>
<p>The World Wide Web, abbreviated as WWW and
commonly known as the Web, is a system of
interlinked hypertext documents accessed via the
Internet.</p>
<p>With a web browser, one can view web pages
that may contain text, images, videos, and other
multimedia and navigate between them by using
```

```
hyperlinks.</p>
</body>
</html>
```

g. Add one more paragraph and a bulleted list as shown.

```
<html>
<head>
<title>Web Page Project 1</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>
<h1>The World Wide Web</h1>
<p>The World Wide Web, abbreviated as WWW and
commonly known as the Web, is a system of
interlinked hypertext documents accessed via the
Internet.</p>
<p>With a web browser, one can view web pages
that may contain text, images, videos, and other
multimedia and navigate between them by using
hyperlinks.</p>
<p>There are many different types of web sites,
including:</p>
<ul>
<li>Social networking</li>
<li>Publishing</li>
<li>Wikis</li>
<li>Shopping and catalog</li>
<li>Search portal</li>
</ul>
</body>
</html>
```

h. Add a comment at the bottom of the page that includes your name as shown. Comments do not appear in the browser window.

```
<html>
<head>
<title>Web Page Project 1</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
</head>
<body>
<h1>The World Wide Web</h1>
<p>The World Wide Web, abbreviated as WWW and
commonly known as the Web, is a system of
interlinked hypertext documents accessed via the
Internet.</p>
```

```
<p>With a web browser, one can view web pages
that may contain text, images, videos, and other
multimedia and navigate between them by using
hyperlinks.</p>
<p>There are many different types of web sites,
including:</p>
<ul>
<li>Social networking</li>
<li>Publishing</li>
<li>Wikis</li>
<li>Shopping and catalog</li>
<li>Search portal</li>
</ul>
<!-- Web page project #1 by Your Name -->
</body>
</html>
```

i. Save your file and view it in your browser. It should look like Figure 1-13.

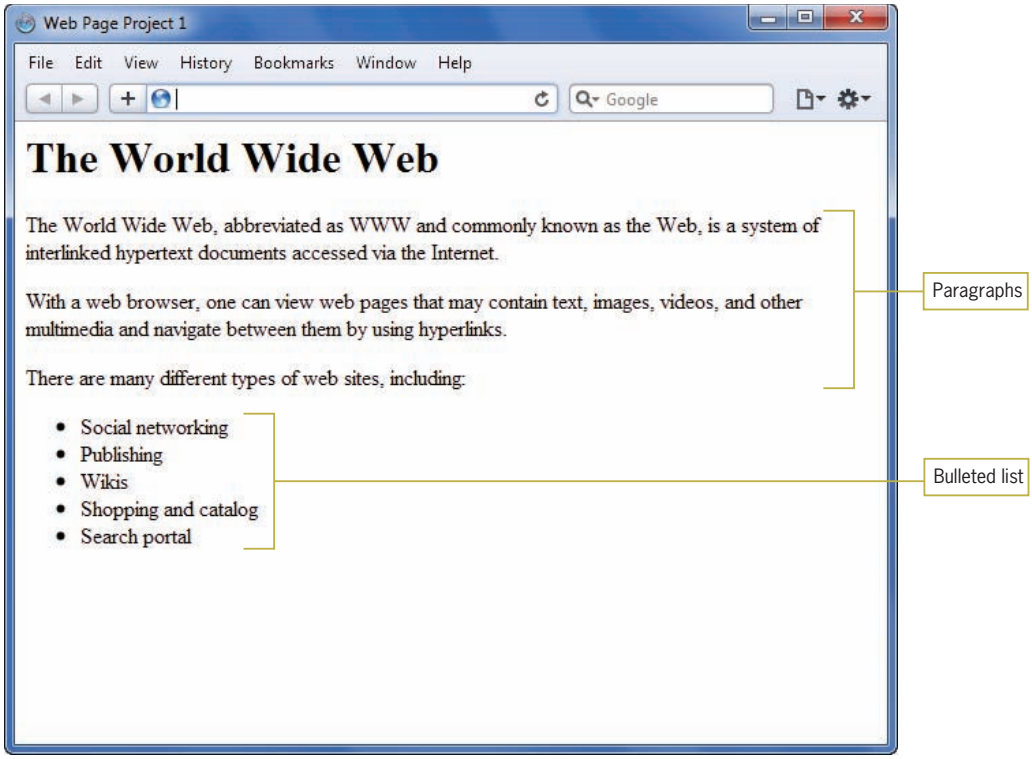


Figure 1-13 Completed project1.htm file

3. Add CSS style rules to a basic HTML file and test it in the browser.

   a. Create a copy of the **project1.htm** file and rename it **project2.htm**.

   b. In an HTML editor, open the **project2.htm** file. Change the title to Web Page Project 2. Change the comment at the end of the document to Web page project #2 by Your Name. Save the file.

   c. Open the **project2.htm** file in your browser. It will look Figure 1-14.

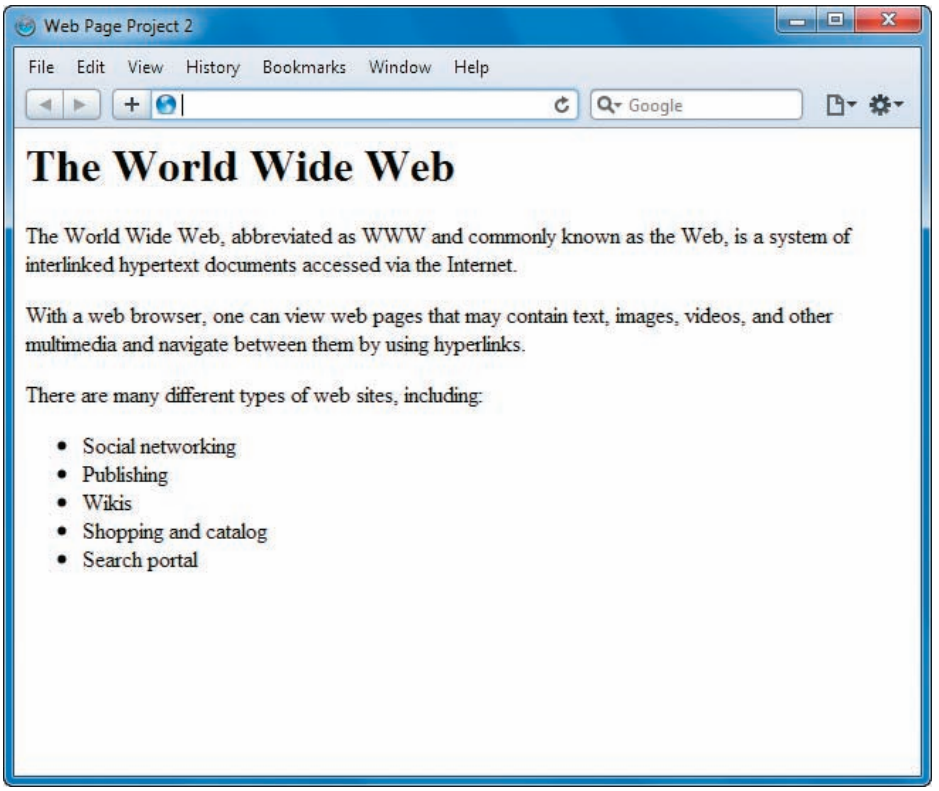


Figure 1-14 Beginning project2.htm file

d. Return to the HTML editor and add a <style> element in the <head> section to contain your style rules, as shown in color in the following code. Leave a line or two of white space between the <style> tags to contain the style rules.

```
<html>
<head>
<title>Web Page Project 2</title>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<style type="text/css">

</style>
</head>
<body>
<h1>The World Wide Web</h1>
<p>The World Wide Web, abbreviated as WWW and
commonly known as the Web, is a system of
interlinked hypertext documents accessed via the
Internet.</p>
<p>With a web browser, one can view web pages
that may contain text, images, videos, and other
multimedia and navigate between them by using
hyperlinks.</p>
<p>There are many different types of web sites,
including:</p>
<ul>
<li>Social networking</li>
<li>Publishing</li>
<li>Wikis</li>
<li>Shopping and catalog</li>
<li>Search portal</li>
</ul>
<!-- Web page project #1 by Your Name -->
</body>
</html>
```

e. Add a style rule that sets the font-family for the page as shown below. Notice the style rule contains both a specific style (Arial) and a fallback style (sans-serif) in case the user does not have Arial as an installed font.

```
<style type="text/css">
body {font-family: arial, sans-serif;}
</style>
```

f. Write another style rule that adds a solid thin bottom border to the <h1> heading element.

```
<style type="text/css">
body {font-family: arial, sans-serif;}
h1 {bottom-border: solid thin;}
</style>
```

g. Save your file and view it in your browser. It should look like Figure 1-15.

Figure 1-15 Completed project2.htm file

4. Visit the World Wide Web Consortium Web site, and find the Web-based validator at *validator.w3.org/unicorn*. Validate the code from the previous two exercises.

5. Convert a text document to HTML from scratch. Open a text file, save it as an HTML file, and then add all markup elements to create a basic Web page. View the browser result, and then validate the file.

6. Download and install at least two different HTML-editing programs that have code-based editing abilities. Write a short report detailing the capabilities of each. Create an HTML document that contains your findings and is viewable in the browser.

Individual Case Project

To complete the ongoing Individual Case Project for this book, you must create a complete stand-alone Web site. The site must have 6–10 pages that display at least three levels of information. You can choose your own content. For example, you can focus on a work-related topic, create a personal interest site, or design a site for your favorite nonprofit organization. The site will be evaluated for cohesiveness, accessibility, compliance with W3C standards, and visual design. At the end of each chapter, you will complete a different section of the project. For Chapter 1, get started by creating a project proposal, using the following outline. As you progress through the chapters of the book, you will complete different facets of the Web site construction, resulting in a complete Web site.

Project Proposal

Create a one- or two-page HTML document stating the basic elements you will include in your Web site. Create this document using your favorite HTML editor or Notepad. At this stage, your proposal is primarily a draft. At the end of Chapter 2, you will have a chance to modify the proposal and supplement the design details.

Include the following items, if applicable:

- *Site title*—Specify the working title for the site.
- *Developer*—Identify yourself and anyone else who will work on the site.
- *Rationale or focus*—Explain the content and goals of the site, such as billboard, customer support, catalog/e-commerce, informational, or resource. Refer to Chapter 3, "Planning the Site," for help on content types.
- *Main elements outline*—Describe the main features of the site.
- *Content*—Estimate the number of individual Web pages.

- *Target audience*—Describe the typical audience for the site.
- *Design considerations*—List the design goals for the site.
- *Limiting factors*—Identify the technical or audience factors that could limit the design goals of the site.

Team Case Project

To complete the ongoing Team Case Project for this book, you and your team must create a complete stand-alone Web site. Your team should ideally consist of 3–4 members assigned by your instructor. The site must contain between 16 and 20 pages that display at least three levels of information. You will choose your own topic. For example, you can focus on a work-related topic, create a personal interest site, or develop a site for a fictional organization. The site will be evaluated for cohesiveness, accessibility, compliance with W3C standards, and visual design. At the end of each chapter, you will complete a different section of the project. For Chapter 1, get started by creating a project proposal, using the following outline. As you progress through the remaining chapters of the book, you will complete different facets of the Web site construction, resulting in a complete Web site.

Project Proposal

Collaborate to create a one- or two-page HTML document stating the basic elements you will include in your Web site. Create this document using your favorite HTML editor or Notepad. At this stage, your proposal is primarily a draft. At the end of Chapter 2, you will have a chance to modify the proposal and supplement the design details.

Include the following items, if applicable:

- *Site title*—Specify the working title for the site.
- *Development roles*—Identify each team member and individual responsibilities for the project.
- *Need*—Describe the need the Web site will satisfy. What is the purpose of the site? Is there an interest group whose needs are not satisfied? Is there a target niche you are trying to fill?
- *Rationale or focus*—Explain the content and goals of the site such as billboard, customer support, catalog/e-commerce, informational, or resource. Refer to Chapter 3, "Planning the Site," for help on content types.

- *Main elements outline*—Describe the main features of the site.
- *Content*—Estimate the number of individual Web pages.
- *Target audience*—Describe the typical audience for the site.
- *Design considerations*—List the design goals for the site.
- *Limiting factors*—Identify the technical or audience factors that could limit the design goals of the site.
- *Development schedule, milestones, and deliverables*—Using the dates in your class syllabus as a basis, build a development schedule that indicates milestones and deliverables for each team member.

CHAPTER 2

Web Site Design Principles

When you complete this chapter, you will be able to:

- Understand the Web design environment
- Design for multiple screen resolutions
- Craft the look and feel of the site
- Create a unified site design
- Design for the user
- Design for accessibility

This chapter covers the design principles that you will apply to your Web page design as you work through this book. By examining current Web design theories and viewing a variety of Web sites, you learn to focus on both the user's needs and the requirements of the content you want to deliver.

The sample Web pages in this chapter come from a wide range of sites. The Web is so far-reaching in content and design that no collection of pages can represent what is typical. Most of the samples illustrate good design principles, although some contain design defects as well. In truth, almost every site has one flaw or another, whether it is a confusing interface, overambitious design, or poor accessibility. Judge the samples with a critical eye. Look for elements of design that you can transfer to your own work. As you progress through the book, you will practice and apply these principles to your own Web design efforts.

Understanding the Web Design Environment

In this section, you will learn about the external factors that affect your Web design efforts. Even though Web coding and design standards have progressed significantly in recent years, many variables still affect how your Web page designs appear to users. As browsers have become more standardized, other more recent factors continue to change and pose challenges for Web designers. These challenges include new screen resolutions based on popular wide-screen monitor formats and new devices such as the iPhone and other handheld devices and e-readers. At the same time, not all users have the latest technology or fastest Internet access. You do not want to leave these users behind.

To be successful, your Web site design must be portable and accessible by users who have a variety of browsers, operating systems, device platforms, and physical abilities. Many designers make the mistake of testing in only one environment, assuming that their pages look the same to all of their users. No matter how much Web design experience you gain, always remember to test in different environments and with different users, even when you feel confident of your results.

You can avoid portability problems by coding to standards as described in Chapter 1 and testing for compatibility. Viewing your pages in multiple browsers, testing on the available operating

systems, viewing on all possible devices, and using different input devices ensure that your site is accessible to the greatest number of users. Consider analyzing your audience and building a profile of your average user. You will read more about analyzing your audience in Chapter 3.

Browser Compatibility Issues

One of the greatest challenges facing HTML authors is designing pages that multiple browsers can display properly. As discussed in Chapter 1, every browser contains a program called a rendering engine that interprets the markup tags in an HTML file and displays the results in the browser. The logic for interpreting the HTML tags varies from browser to browser, resulting in potentially conflicting interpretations of the way the HTML file is displayed. As a Web page designer, you must test your work in as many browsers as possible to ensure that the work you create appears as you designed it. You might be surprised to see that the results of your HTML code can look different when viewed with various browsers.

Often, HTML authors do not have the luxury of knowing the user's operating system or the age and type of browser that will be used to view their Web pages. Browser and version choices can vary widely based on a number of variables. Many people and organizations are reluctant to upgrade software simply because a new version has been released. Although it is a good idea to test with the latest browsers, it also is prudent to test your work in older browsers when possible to maximize the number of people your Web pages can reach.

You can find complete browser compatibility charts at *www.quirksmode.org/compatibility.html*.

You may never be able to achieve the exact same look across all the browsers that are available, but you should try to minimize differences as much as possible so that the greatest number of users experiences your design as you intended. The more you work with HTML and CSS, the more you will realize that slight differences inevitably occur from browser to browser that may not matter to the user. The advances in browser technologies and their adherence to standards, combined with the greater acceptance of standards-based design, now make it easier to build well-designed sites that are displayed consistently from one browser to another.

To download a particular browser, or find out which browser is currently the most popular, visit one of these Web sites:

- BrowserNews at *www.upsdell.com/BrowserNews*
- CNET Browser Info at *www.browsers.com*
- Internet Browser Review at *http://internet-browser-review.toptenreviews.com*
- Netmarketshare has the latest browser market share statistics at *http://marketshare.hitslink.com*

As discussed earlier, not only are new browsers released frequently, but older browsers are still used by many Web users. Including newly supported features in your page design may significantly affect the way your page is viewed if the older browsers cannot interpret the latest enhancements. Browsers exhibit subtle differences across computing platforms as well.

The newer browsers offer much better support for the standards released by the W3C. Browser software companies have found that, as part of the Web development community, they can benefit from the increased support of the standards. Standards-compliant browsers allow better visual design and a more consistent experience for all users.

To ensure the greatest compatibilities of your Web pages across multiple browsers, follow these guidelines:

- *Follow W3C standards*—Use HTML and CSS correctly and consistently.
- *Validate your code*—Test for syntactical correctness and coding errors.
- *Know your audience*—Create designs that are accessible, readable, and legible.
- *Test your work in multiple browsers and devices*—Test and retest as you develop to mitigate problems as they occur.

Connection Speed Differences

The user's Internet connection speed has traditionally been a variable that Web designers could not ignore. If pages download slowly because they contain large, detailed graphics or complicated Flash animations, users may click to go to another site before they see even a portion of content. In the United States, the number of people who can use broadband access to the Web is increasing, which makes connection

speeds less of an issue. According to Websiteoptimization.com (*www.Websiteoptimization.com/bw/1002*), U.S. broadband penetration grew to 95.11 percent among active Internet users in January 2010. Dial-up users connecting at 56 Kbps or less now make up 4.89 percent of active Internet users.

If you are building Web sites for a worldwide audience, you still must plan your pages so they are accessible at a variety of connection speeds. Figure 2-1 shows that in many countries, broadband access is not universal. iPhones and other cellular handsets that display Web pages often have lower bandwidth than home or business computers. You want to consider all possible users when you design the look and feel of your site. Many designers make the mistake of not testing their pages at different connection speeds. If you do not test, you cannot appreciate what it is like for users to connect at different speeds to your site, and you may lose valuable visitors.

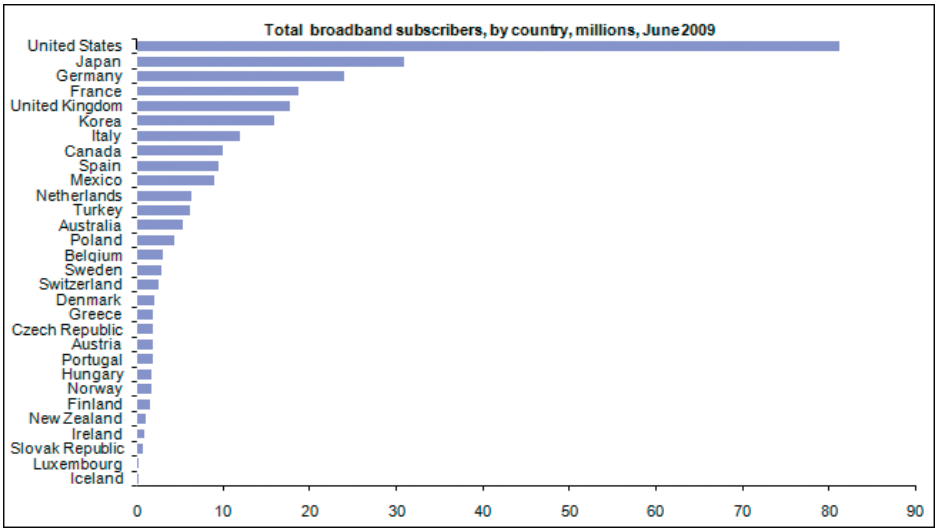


Figure 2-1 Total number of broadband subscribers, by country, millions, June 2009

Browser Cache and Download Time

All Web pages are stored on computers called Web servers. When you type a Uniform Resource Locator (URL) address in your browser, it connects to the appropriate Web server and requests the file you specified. The server serves up the file so your browser can download it. The first time you visit a site, the entire contents of the HTML file (which is plain text), every image referenced in the HTML code, and any CSS style sheets are downloaded to your hard drive. The next time you visit the Web site, your

browser downloads and parses the HTML file from the site. The browser checks to see if it has any of the specified images stored locally on the computer's hard drive in the cache. The **cache** is the browser's temporary storage area for Web pages and images. The browser always tries to load images from the cache rather than downloading them again from the Web.

You can take advantage of the browser's caching capabilities by reusing graphics as much as possible throughout your site. Once an image is downloaded, it remains in your user's cache for the number of days specified in the user's preference settings. Most users do not change the settings, so there is a good chance your graphics will remain on the user's hard drive for as long as a month. Every time the user revisits your site, the cached graphics load locally rather than from the Web server. The browser's caching capability is a great argument for standardizing the look of your site by using the same navigation, branding, and background graphics throughout. Not only does the graphic consistency increase the usability of your site, but your pages also load faster.

As a Web designer, you will be testing and retesting your site. As you do, the browser's caching behavior can work against you. Because the browser stores and reloads files from the cache, your latest changes may not be loaded, especially if filenames match on graphics or style sheets. If you make changes to a Web page, but don't see the results in the browser, it is probably because your browser is reading from the cache rather than loading the changes you made. To clear the cache in most browsers, press the Ctrl+Shift+Del key combination on the PC or hold down the Shift key and click the Reload button in the browser. In Firefox, press Ctrl+F5. Most browsers also let you clear the cache using their menu system. Firefox has an add-on that you can download and install in your browser that adds a Clear Cache button to the toolbar. You can find this add-on at *https://addons.mozilla.org/en-US/firefox/addon/1801*.

Device and Operating System Issues

The user's computer system is the variable over which you have the least control. People use endless combinations of monitors, computers, and operating systems on desktops and mobile devices around the world. The best method for dealing with this variety is to test your content on as many systems as possible, although this may not be realistic for the student or beginning Web designer. Remember the following points about different computer systems:

- *Monitors and display software*—Because of many technical and physical reasons, the colors you choose and images you prepare for your site can look significantly different on different machines. Screen resolutions and sizes, color depth, and video hardware and software all affect the look of your Web pages.

- *Browser versions*—Not all browsers are the same on all operating systems. Often, software companies release new versions of their browsers based on the popularity of the operating system or competition between vendors. For example, Microsoft Internet Explorer is only available on the PC, while Apple makes both a PC and Apple version of its browser, Safari. Other popular browsers such as Firefox and Opera are available for all operating systems. Always test your work in as many browsers as possible.
- *Font choices*—Installed fonts vary widely from one computer to another. Choose fonts that are commonly used; otherwise, if a font you choose is not installed on the user's machine, it will appear in a default typeface. Chapter 5 provides more information on Web typography.

Designing for Multiple Screen Resolutions

No matter how carefully you design pages, you can never know how users view your work because you do not know the screen resolution of their monitors. A computer monitor's **screen resolution** is the width and height of the computer screen in pixels. Most monitors can be set to at least two resolutions, whereas larger monitors have a broader range from which to choose. User screen resolution is a factor over which you have no control.

Screen resolution is a function of the monitor's capabilities and the computer's video card. The current most common base screen resolution (traditionally expressed as width × height in pixels) is 1024 × 768, but this is rapidly being replaced by larger monitors that display at 1280 × 1024, and the popular wide-screen format monitors that display in resolutions of 1200 × 800, 1366 × 768, 1440 × 900, and others. The wide-screen computer displays, both on desktops and laptops, have increased in popularity because they match the wide-screen format of DVD movies. The difference in screen shape between traditional and wide-screen formats affects Web design.

Table 2-1 shows common screen resolutions and the percentage of people using that resolution according to NetMarketShareStats (*http://marketshare.hitslink.com*) as of February 2010.

Resolution (pixels)	Percentage of Users (%)
1024 × 768	26.76
1280 × 800	19.98
1280 × 1024	10.46
1440 × 900	8.81
1680 × 1050	5.53
1366 × 768	4.69
Other	22.36

Table 2-1 Popularity of Screen Resolutions

As shown in Table 2-1, wide-screen formats are gaining a significant share of users. These monitors are now common as new computers are sold with multimedia capabilities and the ability to display movies in a wide-screen format.

Wide-Screen Displays

The new wider screen real estate has changed the way designers are building their page layouts, moving away from flexible layouts that can adapt to different screen resolutions to fixed layouts that display pages consistently no matter the user's resolution. Designers are moving to fixed layouts because of the extended landscape shape of the newer monitors. If a Web browser is maximized in a wide landscape screen, you must account for a tremendous amount of horizontal layout space in your Web design. Figure 2-2 shows the Amazon.com Web site (*www.amazon.com*) in a 1024 × 768 resolution, and Figure 2-3 shows the same page in 1366 × 768, a typical wide-screen format. This page is designed to be flexible and fill the screen at different screen resolutions. In Figure 2-3, you can see that additional white space fills areas in the layout that are flexible to accommodate the wider screen resolution. On very wide displays, this additional space can become so noticeable that it can detract from the layout of the page.

Figure 2-2 The Amazon Web site at 1024 × 768 resolution

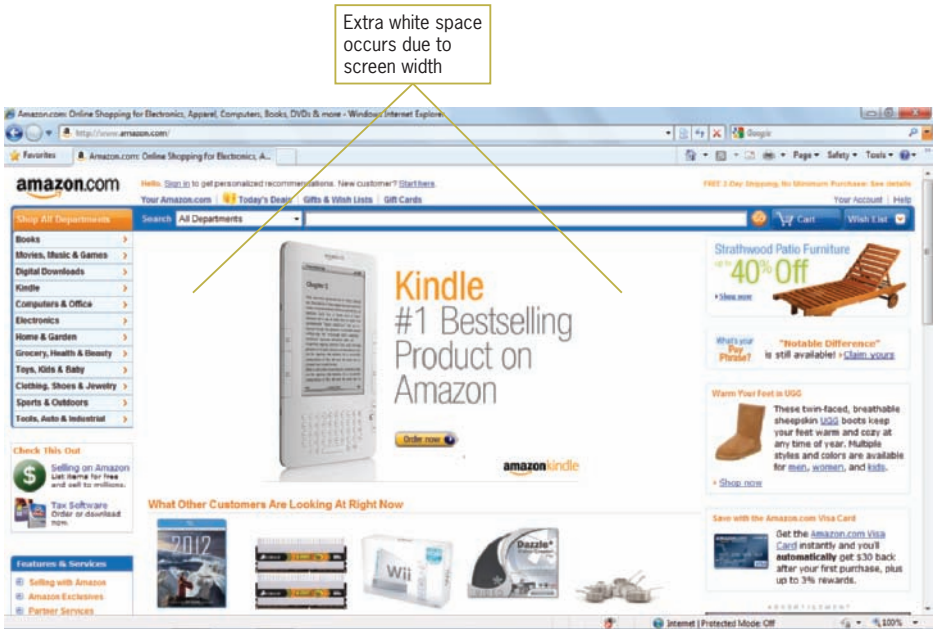


Figure 2-3 The Amazon Web site at 1366 × 768 resolution

Handheld Devices

In addition to wide-screen and standard monitors, many users now have handheld devices such as the iPhone and iTouch that they use for browsing the Web. You should test your Web site designs on these devices as well as desktop and laptop computer monitors. Designing for handheld devices presents a variety of challenges for Web designers including an endless list of protocols and standards, miniscule screen sizes, and inconsistent support for JavaScript applications or Flash animations.

Because people increasingly use handheld devices to access the Web, many Web sites now offer content designed expressly for them. Figure 2-4 shows the main page that Amazon presents for the iPhone, which uses the Apple Safari browser. This page is designed so users can quickly search for an item or check their shopping cart. Amazon.com is serving pages that are appropriate for the user's device type. When an iPhone user enters the Amazon Web address, the Amazon server can detect the iPhone request and respond with the appropriate page.

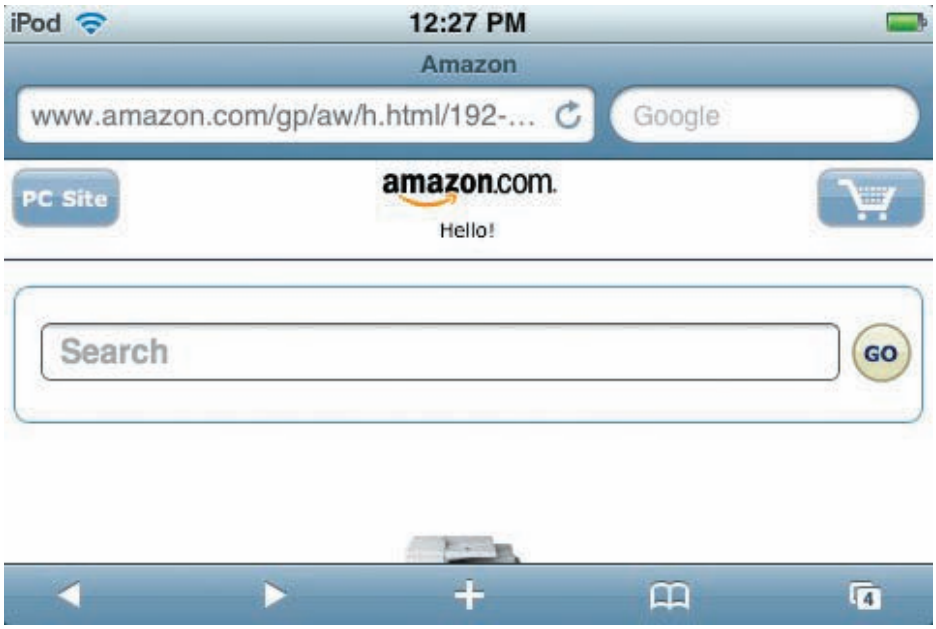


Figure 2-4 Amazon.com home page designed for the iPhone

If iPhone users choose to view the Amazon page designed for personal computers, they see the result shown in Figure 2-5, which, although small, is an exact replica of the Amazon page shown in Figures 2-2 and 2-3. iPhone users spend a lot of time

zooming to view pages that were not intended for their device. Not all Web sites have separate pages designed for the iPhone, but the increasing popularity of the iPhone platform, as well as devices made by other manufacturers, means that this trend will continue.

Figure 2-5 Standard Amazon.com home page viewed on the iPhone

Another method for handling display settings on handheld devices is the CSS handheld media type, which you will learn about in the CSS Media Queries appendix. CSS **Media Queries** (previously called Media Types in CSS2) let you specify style rules for each media destination. The handheld media type lets you provide rules for how your Web page should be displayed on handheld devices.

By specifying a style sheet specifically for handheld devices, you can customize the page layout such as by designing the content to appear in a single column or providing navigation customized for smaller screens. You can also hide elements that will not be displayed properly, such as animated or overly large graphics and JavaScript elements.

You can test your Web site for the iPhone with online emulators such as *http://iphonetester.com* and *www.testiphone.com*, which let you see what your pages look like in the iPhone interface. Better yet, use your own or a friend's iPhone to test your Web pages whenever possible. While emulators are adequate for a quick look at your Web page design, they are not the same as actually trying to navigate through a site using a handheld device.

Flexible Page Layouts

Flexible page layouts, also called fluid layouts, are designed to adapt to different screen resolutions, as demonstrated by the Amazon Web site discussed earlier. Flexible layouts can work especially well and are simpler to design for text-based content. They can pose a variety of design challenges, based on the type of content and complexity of the page layout.

Figures 2-6 and 2-7 show the Wikipedia main page at two different resolutions. Because the Wikipedia content is mainly text, there is minimal visual difference between the two views of the page. The text wraps to fill each column as necessary. A drawback of this design style is that the line of text can be excessively long at wider screen resolutions, decreasing legibility.

Figure 2-6 Using flexible design, content fills window at 1024 × 768 resolution

Figure 2-7 Using flexible design, columns expand to fit at 1366 × 768 resolution

In more complex designs, fluid layout design can be challenging. Your design must account for the movement of elements on the screen as the layout adapts to different browser sizes. For example, the Broads Authority Web site (*www.broads-authority.gov.uk*) has a fluid design that fills the screen at multiple resolutions. Figure 2-8 shows the site at the 1024 × 768 resolution. As the screen expands to accommodate higher resolution, white space is added within the design elements, as shown in Figure 2-9. This technique works well except at extremely high screen resolutions where too much white space could start to break apart the content.

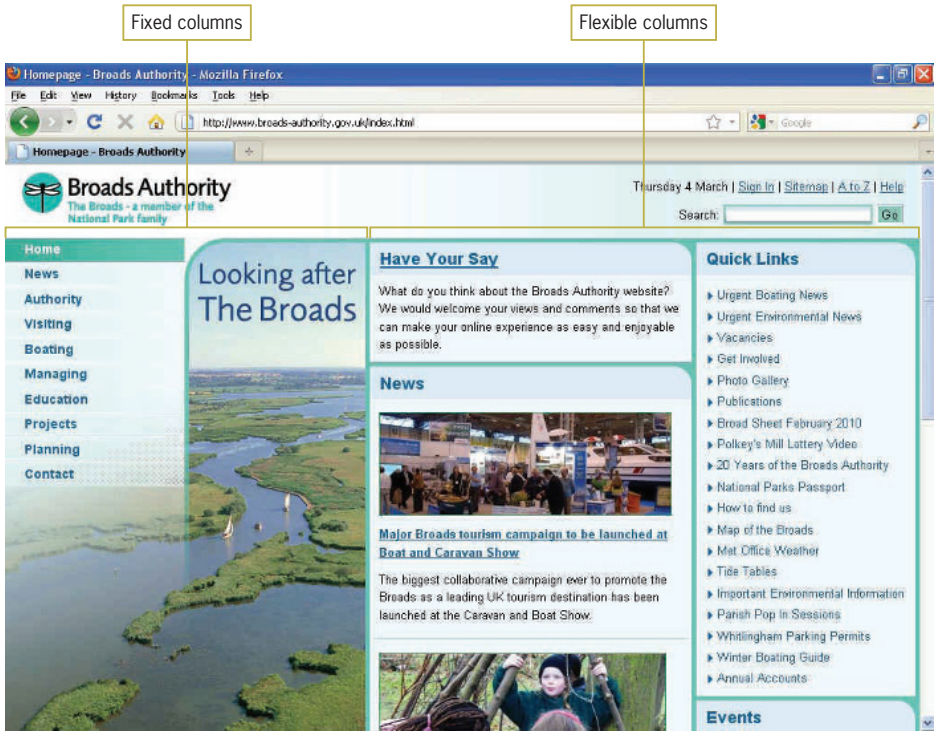


Figure 2-8 Broads Authority Web site at 1024 × 768 resolution

Figure 2-9 Broads Authority Web site at 1366 × 768 resolution

The Broads Authority site is an example of mixed fixed-width and flexible elements. The two left columns that contain the photo and links are fixed, while the right content columns resize to fit the browser window.

Another advantage of flexible design is the ability to adapt to lower screen resolutions. Although users who have their screen resolution set to lower than 1024 × 768 are becoming increasingly scarce, the ability to adapt to lower resolutions and smaller browser windows or device displays means users have ultimate control over how they interact with your content. Figure 2-10 shows the Broads Authority Web site at 800 × 600 resolution. Notice that even at this smaller browser window size, the page content wraps to fit the window, is displayed properly, and does not require the user to scroll horizontally to read the content. Finally, even on the iPhone, the Web site adapts to fit the smaller screen dimensions, as shown in Figure 2-11.

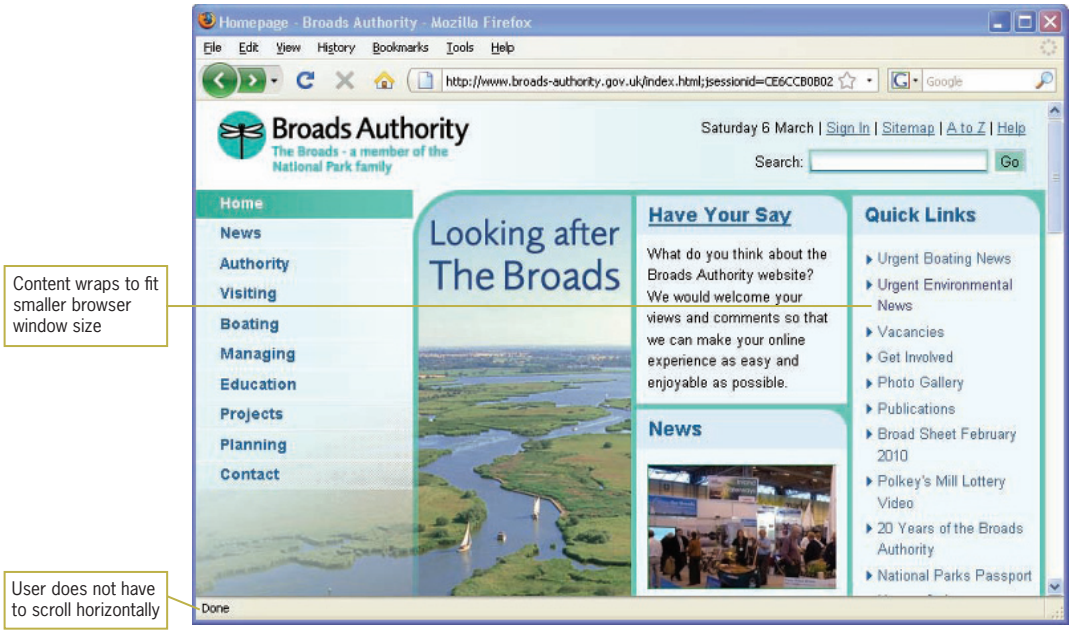


Figure 2-10 Broads Authority Web site at 800 × 600 resolution

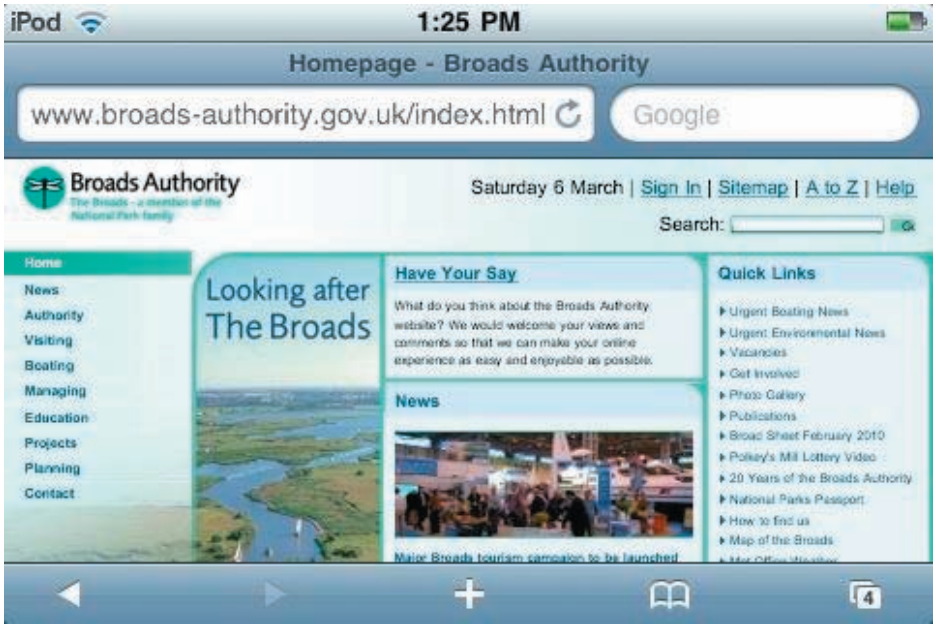


Figure 2-11 Broads Authority Web site on the iPhone

Fixed-Width Page Layouts

Fixed-width page layouts allow the designer to control the look of the Web pages as if it were a printed page, with consistent width and height. Most current fixed-width layouts are designed to stay centered in the browser window, regardless of the user's screen resolution. Fixed-width page layouts have become a popular choice because of the increasing variety of screen sizes and resolutions and the relative ease of constructing fixed-width designs compared to flexible designs. Figures 2-12 and 2-13 show examples of fixed-width designs at different resolutions.

The Boston Vegetarian Society Web site (*www.bostonveg.org*) shown in Figure 2-12 fills the browser window at 1024 × 768 resolution. As the screen resolution changes, the Web page stays centered in the browser window, splitting the remaining space into equal amounts on the left and right side of the browser window. Figure 2-13 shows the same page at 1366 × 768. The benefit of centering a fixed-width page is that the layout of the content remains unchanged no matter what the user's screen resolution.

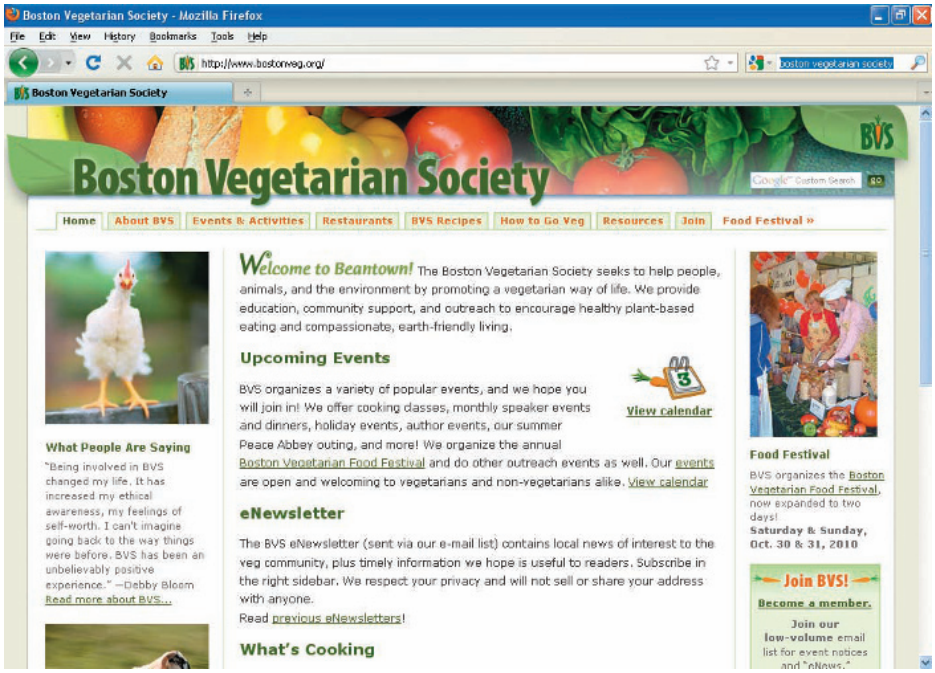


Figure 2-12 Fixed-width design at 1024 × 768 resolution

Figure 2-13 Fixed-width design at 1366 × 768 resolution

Fixed-width layouts increasingly are aligned in the center of the browser window to consistently present the same page format at multiple screen resolutions. Some Web sites still use fixed-width designs that are aligned to the left side of the browser window, as demonstrated by the Washington Post Web site (*www.washingtonpost.com*) in Figure 2-14. This type of design works best for a 1024 × 768 resolution. At higher resolutions, and especially on wide-screen monitors, the browser window displays an abundance of unused space. This figure shows a maximized browser window. Of course, users may choose to resize their browser window so it adapts to the size of the Web site.

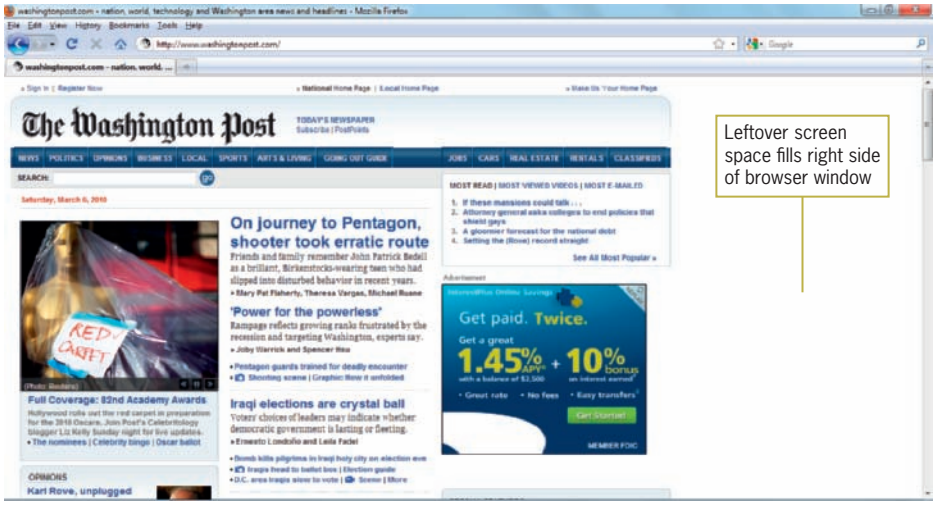


Figure 2-14 Left-aligned fixed-width design at 1366 × 768 screen resolution

Suggestions for Solving the Screen Resolution Dilemma

Which of the two layout types—fixed or flexible—is right for your design? Fixed designs are often chosen for use with Web sites that have more complex page designs, including fixed-size blocks of content, such as video or flash animations. Currently, many more mainstream sites are choosing fixed designs. Flexible designs are more adaptable for the user, but they can have more complicated design requirements. Consider the following advantages of each type of design when you determine which approach is right for you.

Flexible designs:

- User controls the view of the content
- Less chance of horizontal scrolling
- More flexibility for multiple devices
- Better suited to text-based layouts and simpler designs

Fixed-width designs:

- Designer controls the view of the content
- Allows more complex page layouts
- More control over text length

Crafting the Look and Feel of the Site

The interface that the user must navigate is often called the **look and feel** of a Web site. Users look and feel when they explore the information design of your site. They read text, make associations with links, view multimedia, and, depending on the freedom of your design, create their own path through your information. The look and feel is both the way your Web site works and the personality it conveys to the user. Not only should you plan for a deliberate and consistent look and feel, but as mentioned earlier in this chapter, you must test your designs against the variable nature of the Web. You want to ensure that the greatest number of users can navigate your site successfully.

Balance Design and Content

When planning the design of a Web site, access to your content and the needs of your users should always guide your design decisions. As you will read in Chapter 3, Web development teams often comprise many people, each with their own idea of what is important in the current Web site project. Within your company or design team, various stakeholders contribute to the design of the site. The customer has their vision of the finished Web site they are paying your company to design. The designers want to build a site that showcases their design skills. The development team wants to include the latest technologies. The publishing and editorial teams want to highlight their content. Advertising revenues may determine placement and design of ad space on the page.

These varied stakeholders vie for positioning and exposure to their content and content depth, and the larger the site project, the more interests are involved. Everyone wants to contribute their own ideas to the design process. The emphasis on the look of the site can overwhelm the needs of the user, for example, when sites have unnecessary entry pages, too many images, layers that add extra clicks to uncover content, and overdone technology—scrolls, news banners, cycling images, and overdesigned navigation. All of these factors can distract the user from their search for information.

A Web site's design should complement the content and support the reader. The information design should be logically divided and structured to expose similar groupings of content, and then provide access to the content through designed navigation. When in doubt, always choose simple and direct designs that showcase content and allow easy access, and set aside unneeded technology and complex visual designs that can frustrate and misdirect your user.

Plan for Easy Access to Your Information

Your information design is the single most important factor in determining the success of your site. It determines how easily users can access your Web content. The goal is to organize your content and present it as a meaningful, navigable set of information. Your navigation options should present a variety of choices to users without detracting from their quests for information.

A visitor to your site may choose to browse randomly or look for specific information. Often users arrive at a page looking for data that is low in the hierarchy of information. Sometimes users arrive at your site seeking a specific piece of information, such as contact information, product support, or files they want to download. Anticipate and plan for the actions and paths that users are likely to choose when they traverse your site. Provide direct links to the areas of your site that you find or expect to be most in demand. Ask your server administrator for Web server reports to determine which pages are in highest demand, and feature those pages in your design and navigation to allow users easy access. Remember that users want to get to your site, retrieve their desired information, and move on.

Plan for Clear Presentation of Your Information

Even with the current move to higher resolutions and crystal-clear displays, the computer monitor can be a poor reading medium. Environmental factors such as glare or physical distance from the screen can make reading difficult. To counter these problems, design your information so it is easier to read and legible on the screen. Many Web sites fail these criteria by using too many fonts, colors, and lengthy passages of text. Break text into reasonable segments that make for easier on-screen reading. Think about providing contrasting colors that are easy on the eye, such as dark colors against a light or white background. Use plenty of white space to accent specific areas of content and provide separation and structure to your information.

Keep in mind that readers have different habits when reading online. Compared to how they read printed text, online visitors scan more and read less, skimming long pages quickly as they scroll through the text. Include plenty of headings so users can find content quickly. Control the width of your text to provide complete, easy-to-read columns. Keep the "seven (plus or minus two)" rule of information design in mind; that is, users cannot comprehend more than seven (plus or minus two) steps or segments of information at one time. For example, a well-written procedure would contain no more than nine steps. Rather than presenting long scrolling pages, break information into smaller chunks and link them with hypertext.

The Let's Go Web site (*www.letsgo.com*) offers both clear presentation and easy access to information, as shown in Figure 2-15. The navigation links on the left side of the page are hierarchically organized and offer clear descriptions of their destinations. A search text box is located at the top of the page for quick and easy access. The text used throughout the site is legible and easy to read online. Plenty of active white space between the page elements adds to the readability of the page. (You'll learn more about white space later in this chapter.)

Figure 2-15 Clear presentation and easy access

Creating a Unified Site Design

When designing your site, plan the unifying themes and structure that will hold the pages together. Your choices of colors, fonts, graphics, and page layout should communicate a visual theme to users that orients them to your site's content. The theme should reflect the impression that you or your organization want to convey.

When you design a site, you must consider more than each page. For a well-integrated, unified site, plan smooth transitions, use a grid to provide visual structure, and include active white space. Each of these techniques is explained in the following sections.

Plan Smooth Transitions

Plan to create a unified look among the sections and pages of your site. Reinforce the identifying elements of the site and create smooth transitions from one page to another by repeating colors and fonts and by using a page layout that allows you to organize information in a hierarchy. Avoid random, jarring changes in your format, unless this is the effect you want to achieve. Consistency and repetition create smooth transitions from one page to the next, reassuring viewers that they are traveling within the boundaries of your site, and helping them find information.

Provide grounding for the user by placing navigation elements in the same position on each page. Users orient themselves quickly to your navigation structure. Use the same navigation graphics throughout the site to provide consistency and avoid the need to download many graphics.

Think of users turning the pages of a periodical when they browse from Web page to Web page. Although each page should be a complete entity, it also is a part of the whole site. The overall design of a page at any information level should reflect the identity of the site as a whole. For example, Figure 2-16 and Figure 2-17 show the main page and a secondary-level page from the Los Angeles Zoo Web site (*www.lazoo.org*).

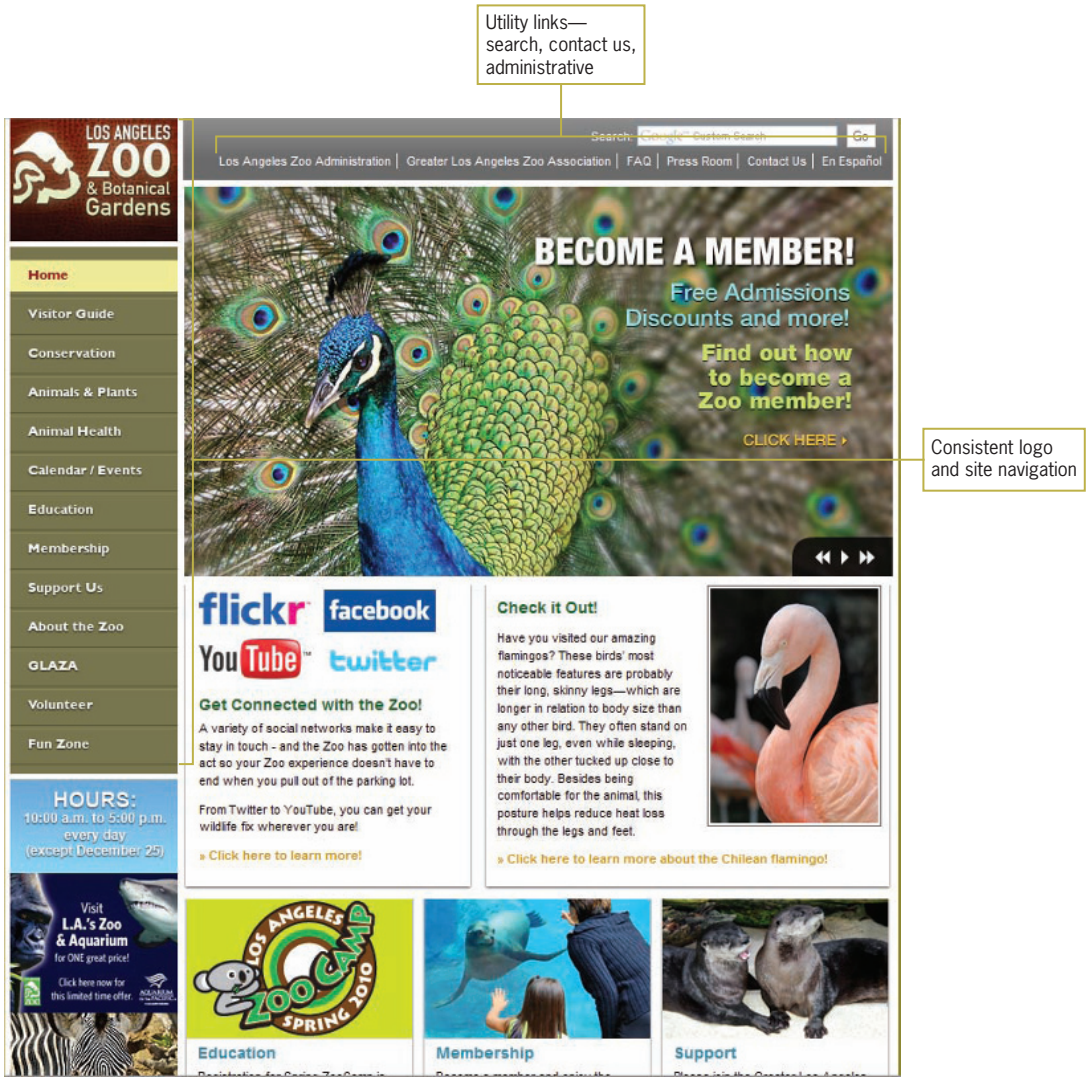


Figure 2-16 Los Angeles Zoo Web site main page

Because these pages share the same color scheme, logo, structure, and navigation, the Web site offers a smooth transition from the main page to the secondary page and presents a unified look and feel.

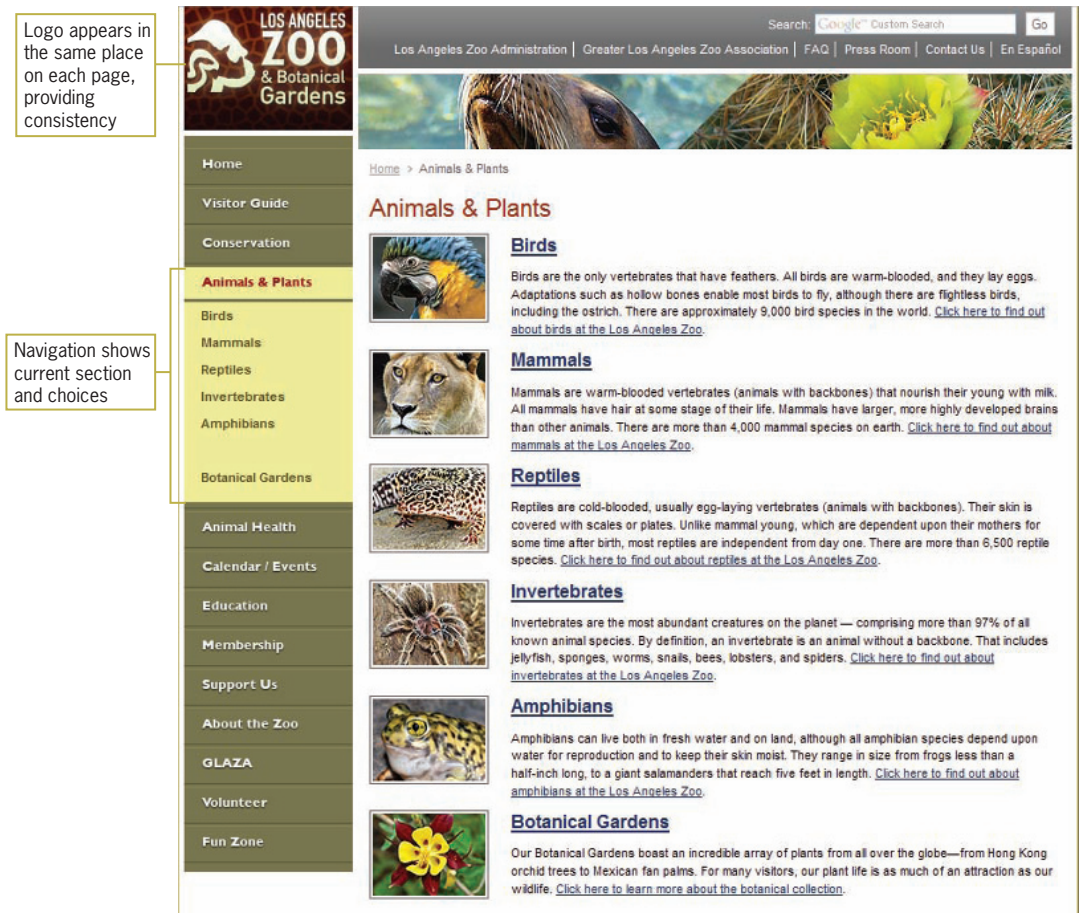


Figure 2-17 Los Angeles Zoo Web site secondary page

Use a Grid to Provide Visual Structure

The structure of a Web page is imposed by the grid or page template you choose for your page design. The **grid** is a conceptual layout device that aligns your page content into columns and rows. You can impose a grid to provide visual consistency throughout your site. You can use the grid to enforce structure,

but you also can break out of the grid to provide variety and highlight important information. Figure 2-18 shows a Web page divided into four columns and eight rows. These grid sections provide placement guidelines for page elements, each of which can cover multiple rows and columns of the grid as needed, as shown in Figure 2-19.

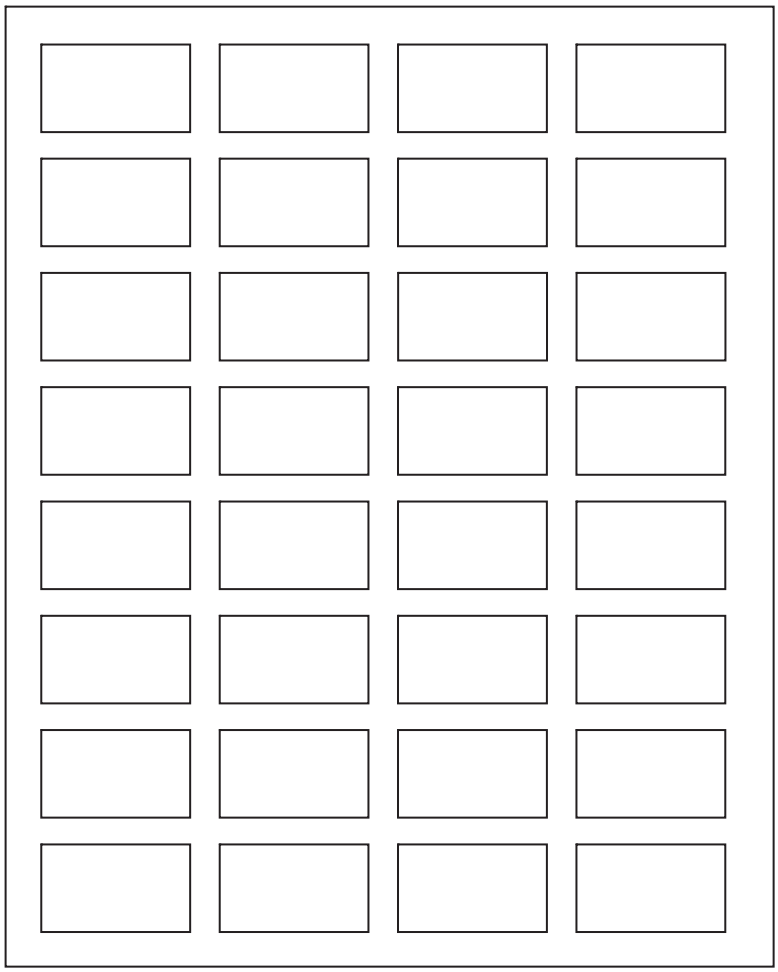


Figure 2-18 Four-column grid

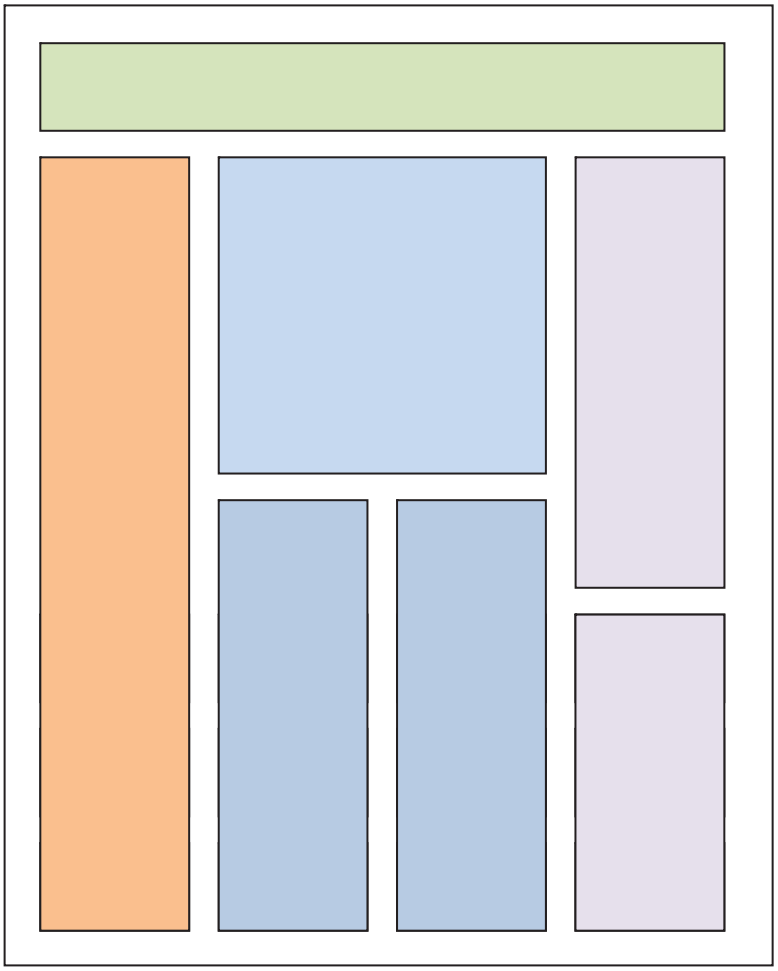


Figure 2-19 Four-column grid with layout elements

Notice that the grid also provides a page margin around the content and gutters of white space between elements on the page. This white space actively separates the content and provides structure for the users' eye to follow. Web pages that respect the grid and consistently align text and graphic elements have a more polished look than pages that have scattered alignments.

The *Guardian* site's main page (*guardian.co.uk*) in Figure 2-20 has a six-column grid. All of the text and graphic elements on the page align within the grid to create an orderly layout.

Figure 2-20 Grid provides visual structure

Use Active White Space

White spaces are the blank areas of a page, regardless of the color you choose to give them. Use white space deliberately in your design, rather than as an afterthought. Good use of white space guides the reader and defines the areas of your page. White space that is used deliberately is called **active white space** and is an integral part of your design because it structures and separates content. Sometimes the strongest part of a design is the active white space. White space is any area of the screen that does not include content, regardless of your background color. **Passive white space** includes the blank areas that border the screen or are the result of mismatched shapes. Figure 2-21 illustrates active versus passive white space.

The 960 grid system (*www.960.gs*) has become a commonly accepted standard for many Web designers. The "960" refers to the width in pixels of the page design, which can easily be divided into multiple columns and fits screen resolutions from 1024 pixels wide and up. The Web site has CSS files that let you build grid page designs that can become the basis for your own page designs.

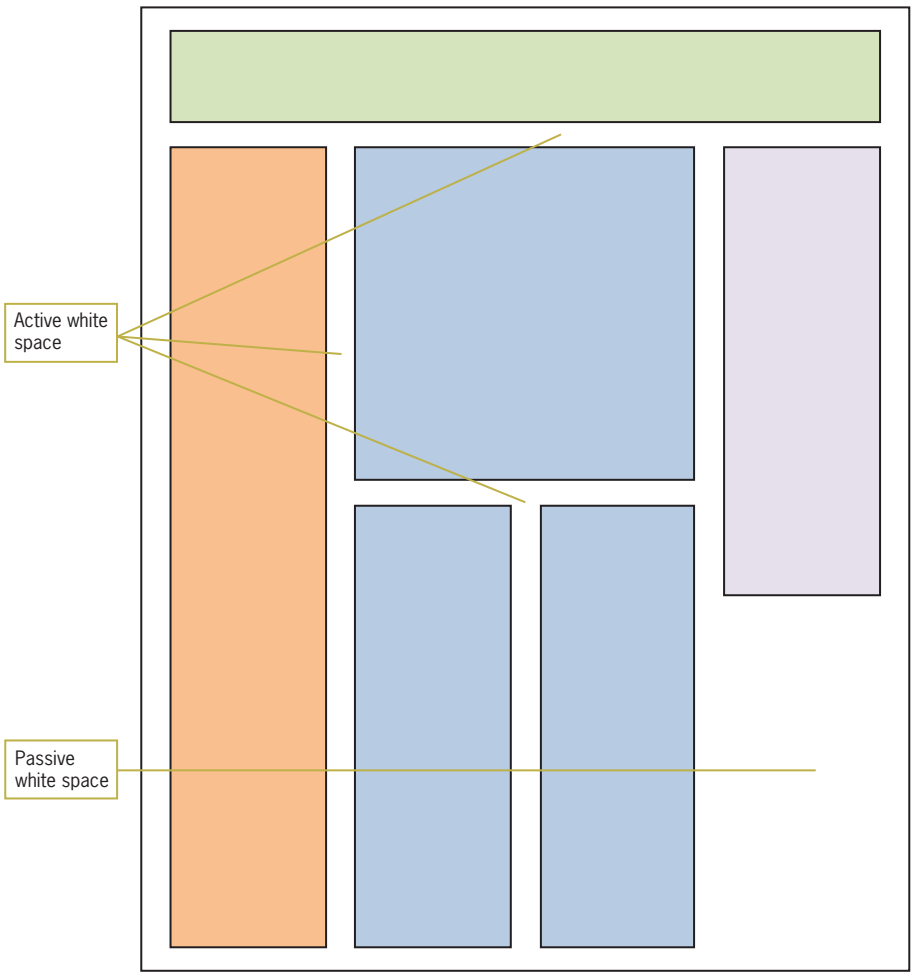


Figure 2-21 Areas of active and passive white space

Content presentation can become confused when designers do not use enough active white space to separate and define content. A lack of active white space creates the impression that a page contains too much information and that it will be difficult to find the piece of information you want. The NASA home page (*www.nasa.gov*) in Figure 2-22 shows effective use of active white space, making the content easy to read. Plenty of active white space reduces clutter and clarifies the organization of your ideas.

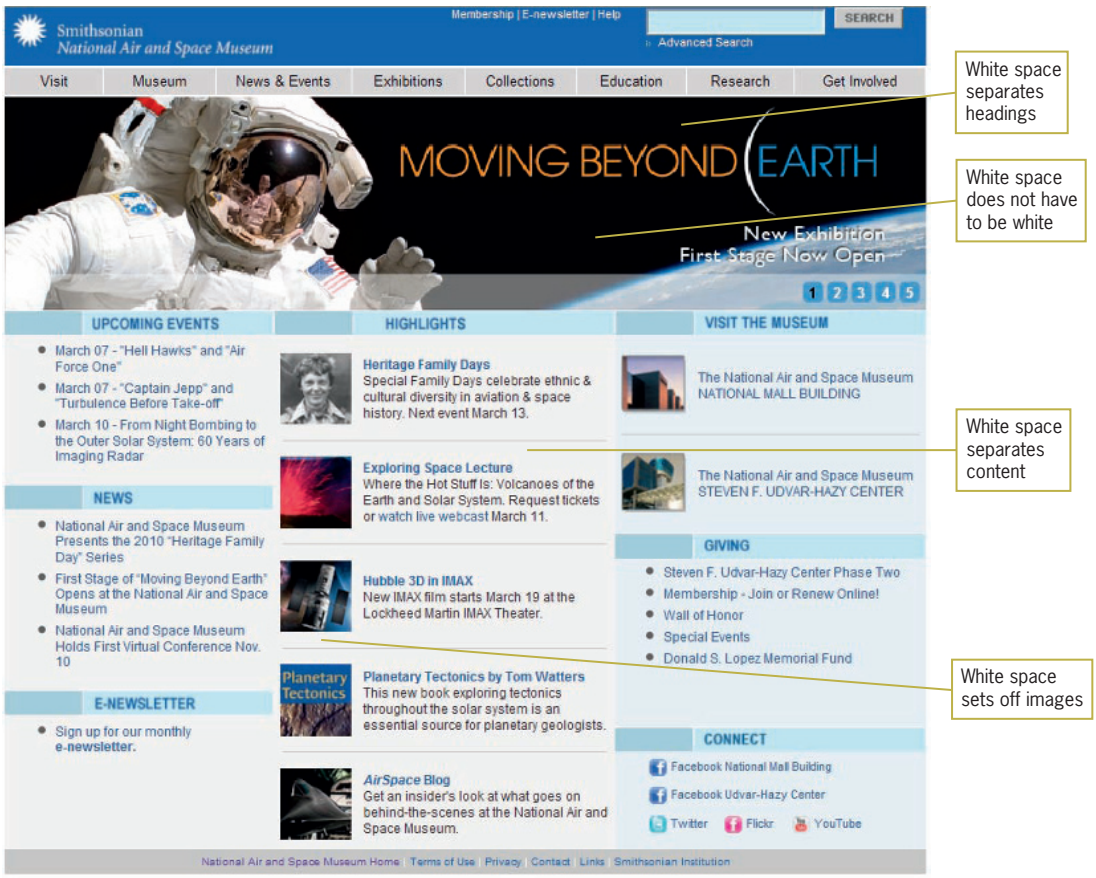


Figure 2-22 Active white space enhances legibility

Designing for the User

Keep your design efforts centered solely on your user. Knowledge of your audience can help you answer almost all design questions—if it serves the audience, keep it; if it is potentially distracting or annoying, eliminate it. Find out what users expect from your site. If you can, survey them with an online form. Create a profile of your average user by compiling responses to basic questions. What do users want when they get to your site? Are they trying to find customer support and troubleshooting help, or do they want to buy something? Do they want to

read articles or search for information? Once you know what your users want from your site, you can evaluate how the design reflects the audience's profile and needs. Consider the main page for Google (*www.google.com*), currently the Web's most popular search engine. The site's main page, shown in Figure 2-23, has no ads, very few links, and is designed for only one purpose—letting users quickly enter a search term.

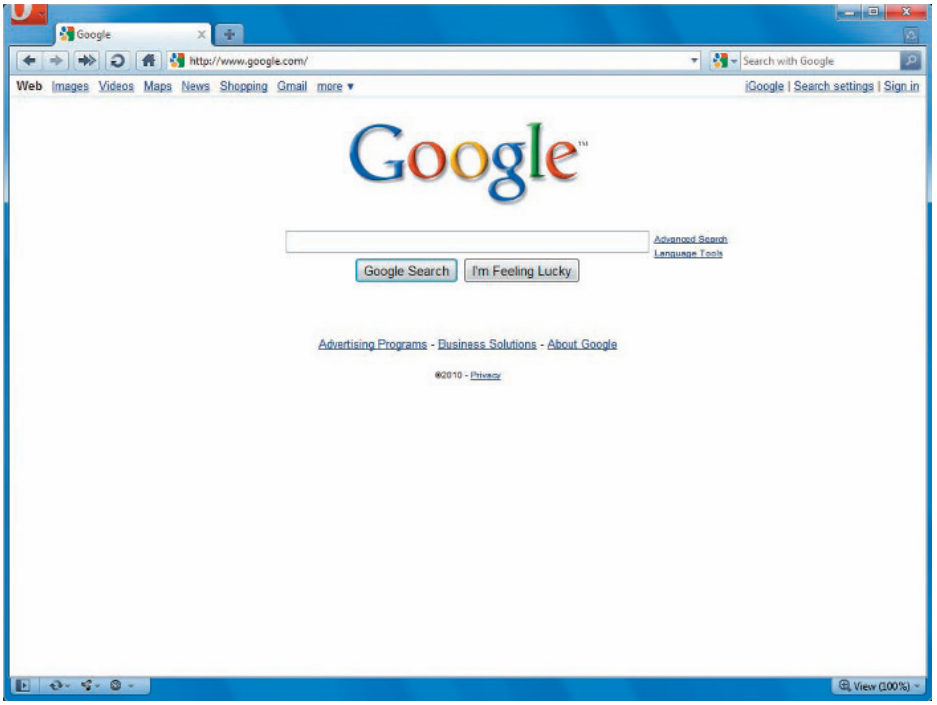


Figure 2-23 Google's simple, task-oriented design

Compare the main pages from the following sites and consider their target audiences. The Cartoon Network Web site (*www.cartoonnetwork.com*) shown in Figure 2-24 contains competing content that draws the user's eye, such as Flash animations, a scrolling navigation bar, bright colors, and familiar shapes. The overall effect is decidedly similar to television cartoons—familiar territory for the site's audience.

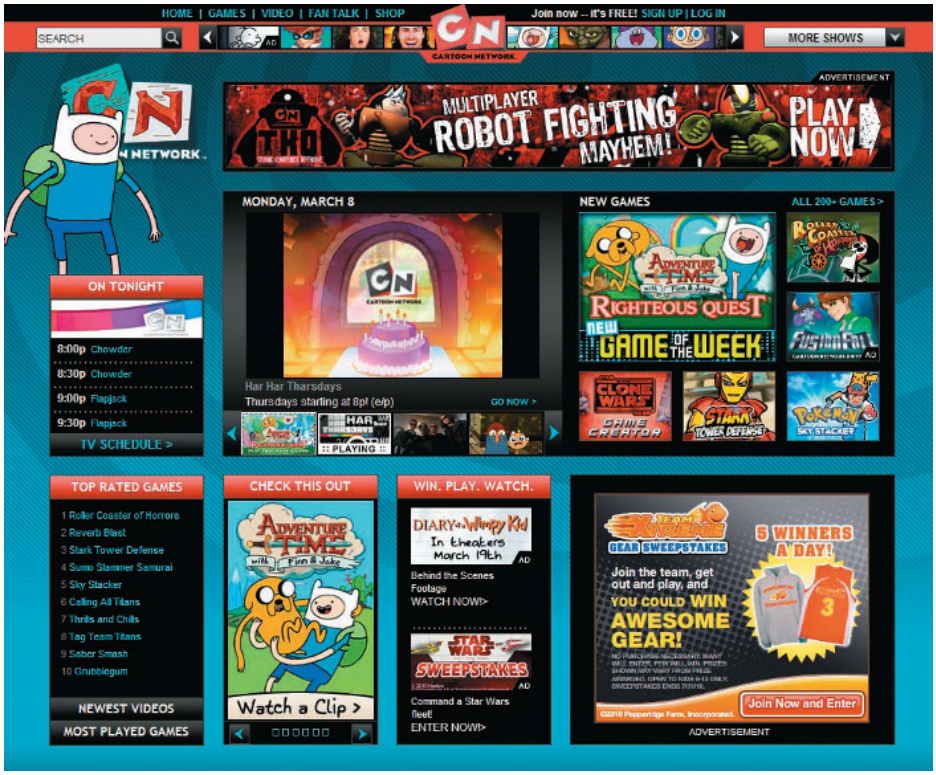


Figure 2-24 Hectic, but appropriate design for Cartoon Network's online audience

In contrast, the Web site for the *The New Yorker* (*www.newyorker.com*) in Figure 2-25 projects a strong periodical-like image. The main page components are textual. Even though the page has a lot of content, it is well organized with clear headings and readable text in well-defined columns. The design uses just enough active white space to clearly separate each element on the page. The overall effect evokes the look of a printed page while using the color, linking, and design flexibility that the Web offers.

Figure 2-25 Paper-based design for *The New Yorker*'s audience

These two examples demonstrate how the design suits the audience's visual expectations—the look of the site. However, you also should consider the ways in which users interact with the content—the feel of the site.

Design for Interaction

Think about how the user wants to interact with the information on your Web page. Design for your content type, and decide whether the user is likely to read or scan your pages.

For example, suppose your page is a collection of links, such as a main page or section page. Users want to interact with these types of pages by scanning the content, scrolling if necessary, pointing to graphics to see if they are hyperlinked, and clicking linked text. Design for this type of user interaction by using meaningful column headings, linked text, and short descriptions. Organize links into related topic groups and separate groupings with white space, graphics, or background color.

Suppose the page is an article that contains large blocks of text. Your user is accustomed to interacting with pages of text by scrolling and possibly clicking hyperlinked words of interest. The links may be in the body of the article or contained in a sidebar. Design your pages for this text-oriented content by keeping paragraphs short for online consumption. Make reading easier by using a text column that is narrower than the width of the screen. Keep your text legible by providing enough contrast between foreground and background colors. Provide links that allow the user to jump quickly to related content.

Two screens from the National Public Radio (NPR) News Web site (*www.npr.org*) illustrate the difference between designing for reading and designing for scanning. For example, Figure 2-26 shows the site's main page. This mainly text-filled page has an open, informational feel. Three columns of content present a variety of information. Users can scan links to find a topic of interest, read article abstracts, or choose one of the featured main sections.

Figure 2-26 Page designed for scanning

When users choose a link, they jump to a page designed for reading, as illustrated in Figure 2-27, which shows a secondary page from the NPR Web site. This page has a two-column layout, with a more generous left column that contains the main article text. Featured main sections are provided in the banner at the top of the page. An article sidebar provides links to related content.

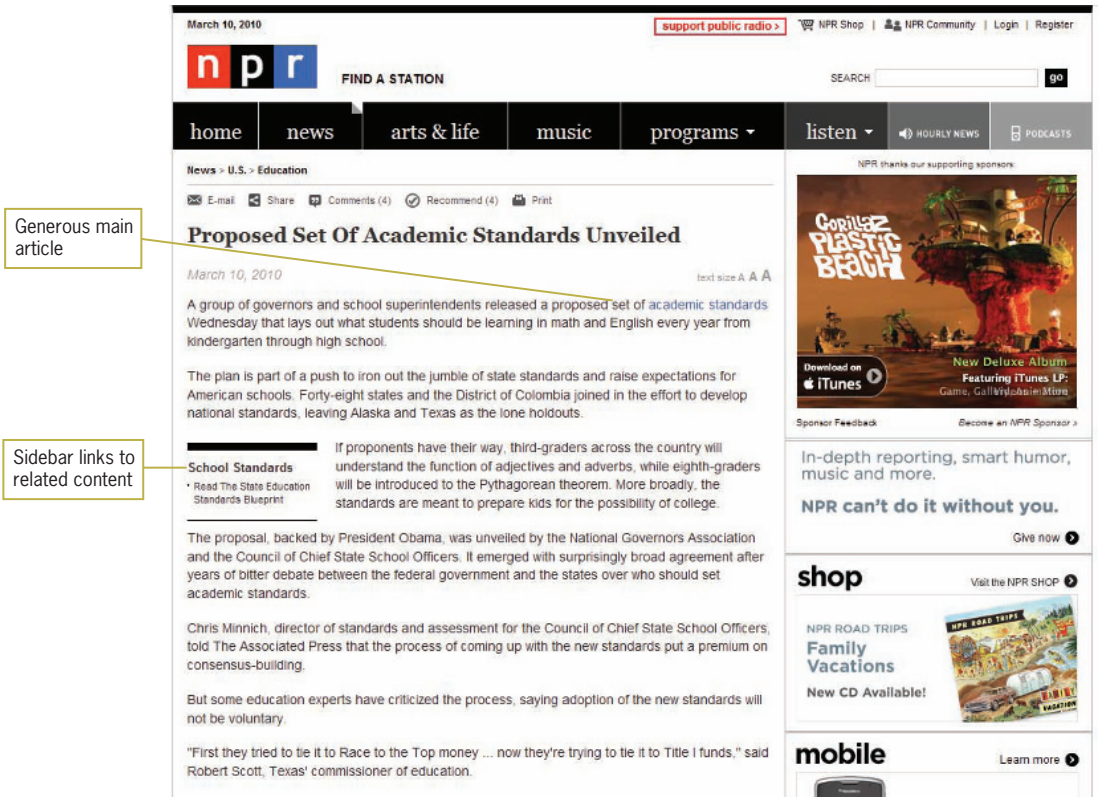


Figure 2-27 Page designed for reading

Design for Location

The user can traverse a page in a variety of ways. Human engineering studies show a wide range of results when tracking a user's eye movements. As you plan your design to guide the user through your content, consider the different ways your user could be viewing your Web pages.

Fixed-width designs tend to have the same proportions as the printed page, which enforces scanning the page using paper-based reading habits. In this reading pattern, the user's eye moves from left to right and back again, as shown in Figure 2-28.

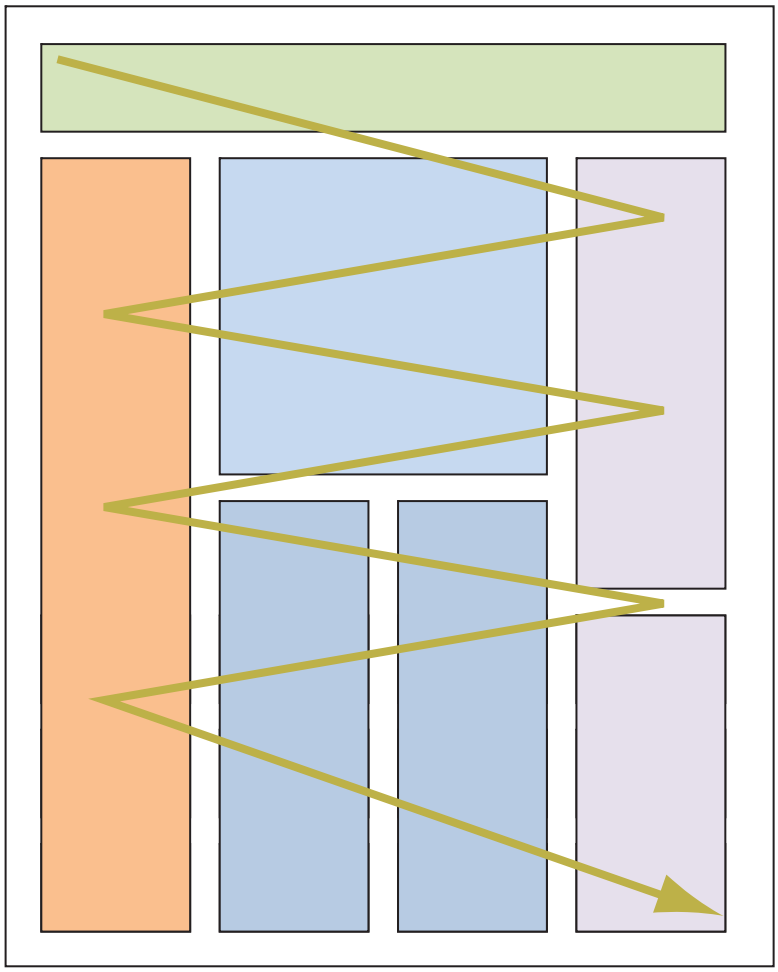


Figure 2-28 Paper-based reading pattern

In contrast, when viewing flexible layouts that fill the screen, users may scan information following a clockwise pattern, as shown in Figure 2-29.

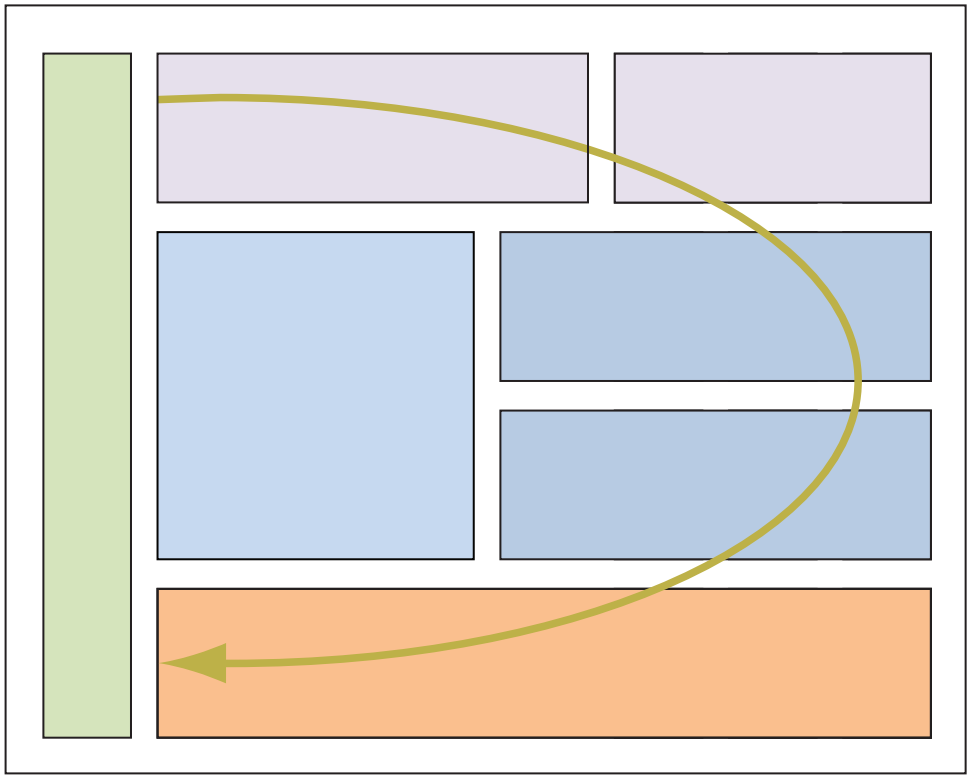


Figure 2-29 Landscape-based viewing pattern

A third pattern, found in eye-tracking studies performed by Jakob Nielsen, shows that "users often read Web pages in an F-shaped pattern: two horizontal scans followed by a vertical scan." (*www.useit.com/alertbox/reading_pattern.html*) The F-shaped pattern, shown in Figure 2-30, is dominated by the upper-left of the page, where users look for the most important information and navigation on the page. According to Nielsen, users' dominant reading patterns tend to follow this sequence:

- Users first read across the top of the page.
- Next, users move down the page a bit and scan across the page again. This scan is typically shorter than the first.
- Finally, users scan the content's left side in a vertical movement. Sometimes this is a fairly slow and systematic scan of the content.

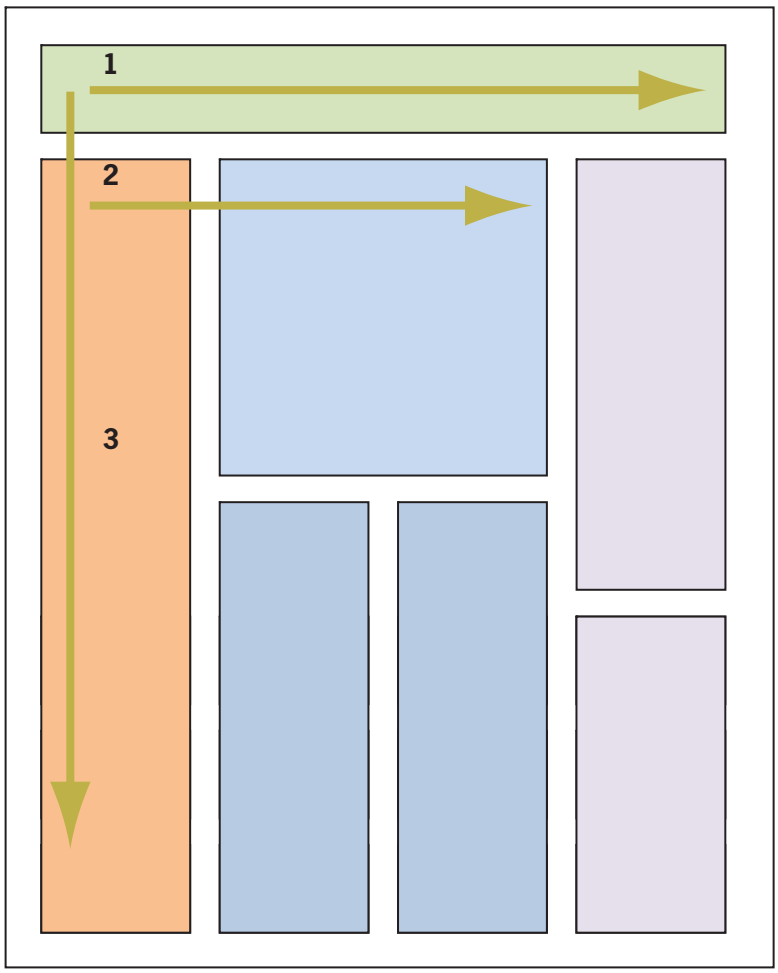


Figure 2-30 F-based viewing pattern

In addition to knowing how the user views your pages, you should also know what expectations he or she might have about your navigation and content. Users have come to expect common elements of a Web page in certain locations. These conventions have evolved because of usage by popular Web sites as well as basic page design criteria. Eye-tracking studies cited by Patrick J. Lynch and Sarah Horton in the *Web Style Guide, Third Edition* (*http://Webstyleguide.com/wsg3/3-information-architecture/4-presenting-information.html*) shows how users, when asked to identify where they expected to find certain elements, more or less agreed on the relative positioning on a Web page grid. In Figure 2-31, the top rows of grids show different element expectations, with the deeper colors indicating higher preference. The page

mockup below the grids shows the aggregate of the results positioned on a Web page design. These expectations reflect many Web sites and would look familiar to anyone who has browsed the Web for any length of time. For example, the Home link graphic shows that most users look to the upper-left corner of the Web page to find a link to the home page. The Shopping cart element shows that most users look to the upper-right or the middle-right to find a shopping cart link. Keep these expectations in mind to satisfy the needs of the user and help solve design problems.

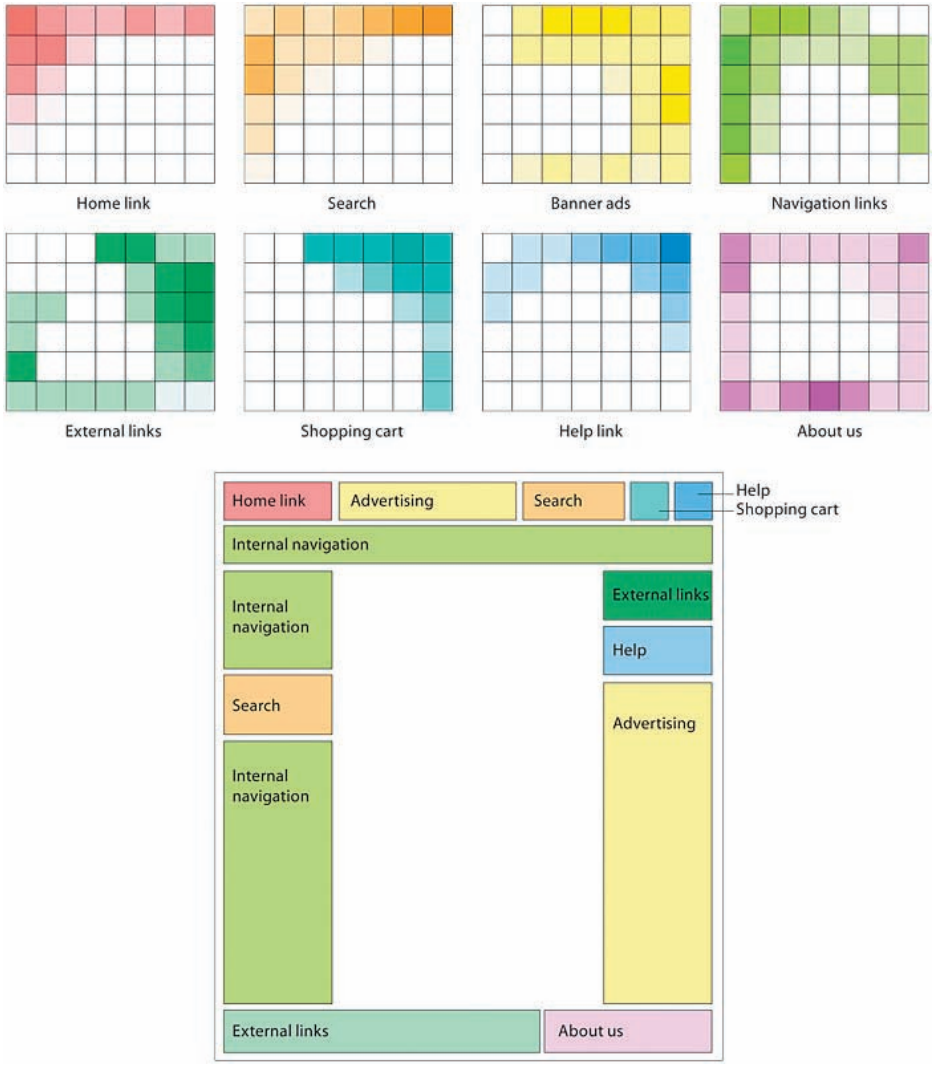


Figure 2-31 User expectations of Web page element locations
(Source: Courtesy of Patrick J. Lynch and Sarah Horton)

Remember that these studies and theories are guidelines for design, not hard and fast rules. They suggest general user tendencies rather than specific habits. Knowing these common user tendencies is one more way to help you decide where to focus the user's attention by object placement, text weight, and color use. As with any good type of information design, keep the following points in mind:

- Think about your grid structure and how you want to break out of it to attract attention.
- Use text weight and size to communicate relative importance of information.
- Use meaningful headings to help users navigate through your content.
- Divide content into sections with rules or active white space.
- Use shapes and color to reinforce location or topic.

Keep a Flat Hierarchy

Do not make users navigate through too many layers of your Web site to find the information they want. Structure your Web site to include section- or topic-level navigation pages that let users find their desired paths quickly. Create content sections organized logically by theme.

Try to follow the three clicks rule; that is, don't make your users click more than three times to get to the content they desire. Provide prominent navigation cues that enable quick access. For example, a standard navigation bar consistently placed on every page reassures users that they will not get lost, and it lets them move through the site with flexibility. Think about the primary tasks the user wants to accomplish at your Web site, and design accordingly to make it easy for the user to accomplish those tasks.

Consider providing a site map that graphically displays the organization of your Web site. Figure 2-32 shows a site map from Google (*www.google.com*). This graphical view of the Web site shows all the individual pages and the sections in which they reside. Clear headings organize the content. Users can click to go directly to a page or orient themselves to the site's structure.

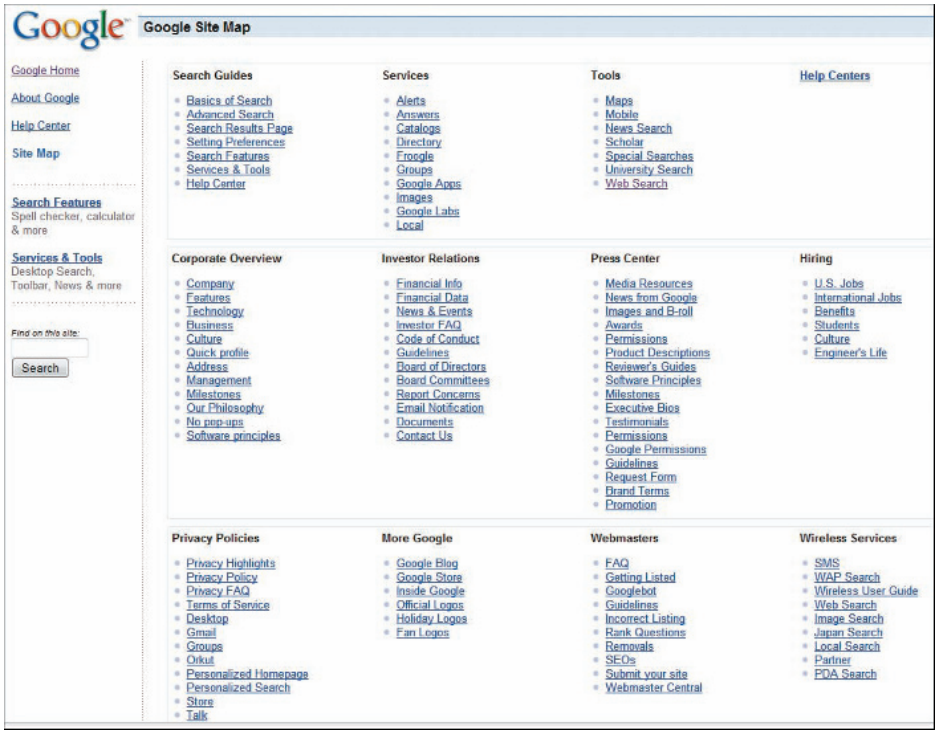


Figure 2-32 Google's site map

Use Hypertext Linking Effectively

Unlike paper-based authors, as an HTML author you have the luxury of adding clickable text and images where necessary to guide users through your information. This powerful ability comes with a measure of responsibility. You make the decisions that determine how users move through your site and process information. Readers browsing through magazines can flip to any page in any order they desire. You can replicate this nonlinear reading method on your Web site with links that let users move from page to page or section to section. With thoughtful hypertext writing, you can engage readers in a whole new way.

Many sites have separate columns of links and topics, but not enough sites provide links within the text. This is a powerful hypertext feature that is not used often enough. Weave links into your prose to offer a variety of paths. Avoid using the meaningless phrase "Click Here" as the hypertext link. Instead provide a helpful textual clue to the destination of the link.

Figure 2-33 shows a page from the Wikipedia Web site (*www.wikipedia.org*). Note how the hypertext links are worked directly into the text. When users click a link, they move to another page of information; from that page they can either go back or move to another page of information, and so on. The abundant hypertext links allow users to create a view of the site's information that is uniquely their own.

Provide plenty of links to let the user get around quickly. Use links to let the user return to the navigation section of your page, to a site map, or to the main page. Do not make the user scroll through lengthy columns. Provide links that let users jump down the page, return to the top of the page, or navigate a clear path back to higher levels of your content.

Figure 2-33 Abundant textual links

How Much Content Is Too Much?

You can crowd only so much information onto a Web page. Be conscious of the cognitive load of the user, who often thinks that Web pages hold too much information. CNBC's Web site (*www.cnbc.com*) in Figure 2-34 offers a dizzying array of content choices.

Figure 2-34 Vast array of choices

Resist the temptation to overload users with too much information. According to the *New York Times*, a study conducted by the University of California San Diego calculated that the average American consumes 34 gigabytes (GB) of content in a single day. That equals thousands of words, images, and multimedia cascading from a variety of sources, the most prominent being television and computers. Users quickly learn to screen out the "noise" of banner and animated ads, images, and blocks of undistinguishable text. To make your content stand out and have impact, carefully divide it into smaller sections and present it in a structured manner with lots of white space and meaningful navigation cues.

Reformat Content for Online Presentation

Although tempting, it often is a poor choice to take documents that are formatted for print and post them online without considering the destination medium. In most cases, a document that is perfectly legible on paper is hard to negotiate online. The text length, font, and content length do not transfer successfully to the computer screen. Figures 2-35 and 2-36 show the same section of text from Edgar Allen Poe's "The Tell-Tale Heart." Figure 2-35 is formatted as if it were a page from a book. The text is dense and fills the screen in large blocks, with no margins to relieve the reader's eye.

In contrast, Figure 2-36, from the Readprint Web site (*www.readprint.com*), shows text that has been designed for online display. The text width is short and easy to read without horizontal scrolling. The font is designed for online reading. The white space on the left creates a text column that enforces the vertical flow of the page. The differences between these two pages show that text must be prepared thoughtfully for online display.

Literature.org
The Online Literature Library

Search Literature.org
Google Search

Literature.org:
Contact

The Tell-Tale Heart

Edgar Allan Poe

TRUE! nervous, very, very dreadfully nervous I had been and am; but why WILL you say that I am mad? The disease had sharpened my senses, not destroyed, not dulled them. Above all was the sense of hearing acute. I heard all things in the heaven and in the earth. I heard many things in hell. How then am I mad? Hearken! and observe how healthily, how calmly, I can tell you the whole story.

It is impossible to say how first the idea entered my brain, but, once conceived, it haunted me day and night. Object there was none. Passion there was none. I loved the old man. He had never wronged me. He had never given me insult. For his gold I had no desire. I think it was his eye! Yes, it was this! One of his eyes resembled that of a vulture -- a pale blue eye with a film over it. Whenever it fell upon me my blood ran cold, and so by degrees, very gradually, I made up my mind to take the life of the old man, and thus rid myself of the eye for ever.

Now this is the point. You fancy me mad. Madmen know nothing. But you should have seen me. You should have seen how wisely I proceeded -- with what caution -- with what foresight, with what dissimulation, I went to work! I was never kinder to the old man than during the whole week before I killed him. And every night about midnight I turned the latch of his door and opened it oh, so gently! And then, when I had made an opening sufficient for my head, I put in a dark lantern all closed, closed so that no light shone out, and then I thrust in my head. Oh, you would have laughed to see how cunningly I thrust it in! I moved it slowly, very, very slowly, so that I might not disturb the old man's sleep. It took me an hour to place my whole head within the opening so far that I could see him as he lay upon his bed. Ha! would a madman have been so wise as this? And then when my head was well in the room I undid the lantern cautiously -- oh, so cautiously -- cautiously (for the hinges creaked), I undid it just so much that a single thin ray fell upon the vulture eye. And this I did for seven long nights, every night just at midnight, but I found the eye always closed, and so it was impossible to do the work, for it was not the old man who vexed me but his Evil Eye. And every morning, when the day broke, I went boldly into the chamber and spoke courageously to him, calling him by name in a hearty tone, and inquiring how he had passed the night. So you see he would have been a very profound old man, indeed , to suspect that every night, just at twelve, I looked in upon him while he slept.

Upon the eighth night I was more than usually cautious in opening the door. A watch's minute hand moves more quickly than did mine. Never before that night had I felt the extent of my own powers, of my sagacity. I could scarcely contain my feelings of triumph. To think that there I was opening the door little by little, and he not even to dream of my secret deeds or thoughts. I fairly chuckled at the idea, and perhaps he heard me, for he moved on

Figure 2-35 Content formatted for print

Figure 2-36 Content formatted for the Web

Designing for Accessibility

Any large audience for a Web site includes users who want to access your content despite certain physical challenges. Designing for accessibility means developing Web pages that remain accessible despite any physical, sensory, and cognitive disabilities; work constraints; or technological barriers on the part of the user. As Tim Berners-Lee said, "The power of the Web is in its universality. Access by everyone, regardless of disability, is an essential aspect." Most mainstream Web sites are so heavily image- and media-intensive that they are not suitable for adaptive devices such as screen readers, voice browsers, and Braille translators.

Many Web sites employ at least some accessibility features, while others are more aggressive about conforming to the standards set by the W3C, which maintains the Web Content Accessibility Guidelines (WCAG) recommendation. Many of these features can be helpful for any visitor to your site. For example, allowing the user to change the font size on a Web page (rather than using the

browser zoom tool) would possibly be used by people with a sight disability and people who have high-resolution monitors where the text appears much smaller. Offering more accessibility features makes your content available to a wider audience.

Building more accessible content does not mean that you have to forgo interesting Web designs. Many of the guidelines necessary for developing accessible content naturally lend themselves to creating good design. Common accessibility features can be unobtrusive additions to your site design. For example, the English in Chester site (Figures 2-37 and 2-38) includes the following among its accessibility features:

- *Optional navigation links*—Lets users with screen readers skip repetitive navigation links and jump directly to the page content
- *High-contrast version*—Lets users switch to a legible alternate page version to make text easier to read
- *User-controlled font size*—Lets users adjust the font size for optimal legibility
- *Access keys*—Let users access sections of the site with keystrokes, which are listed on the Accessibility page.

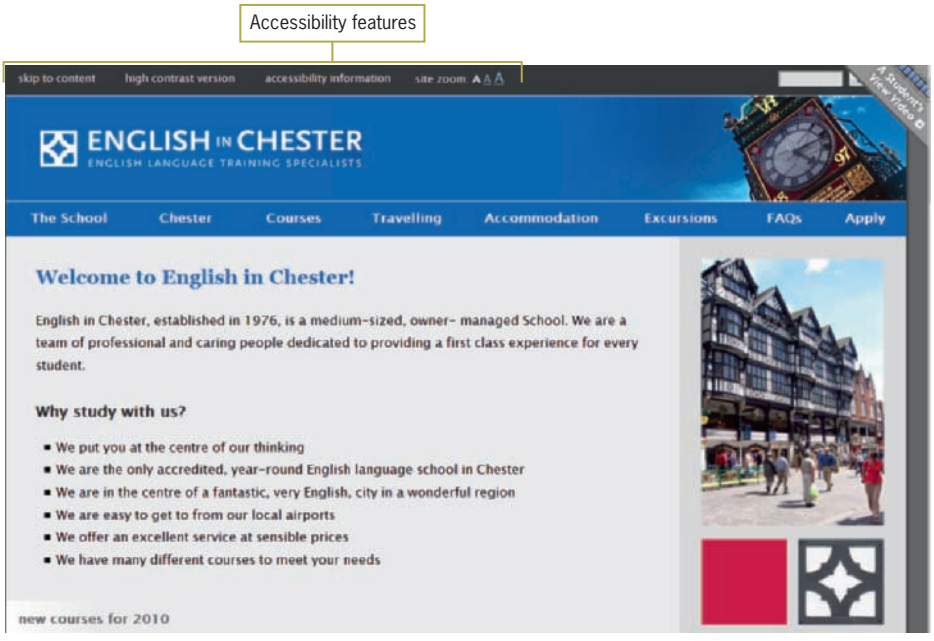


Figure 2-37 English in Chester Web site accessibility features

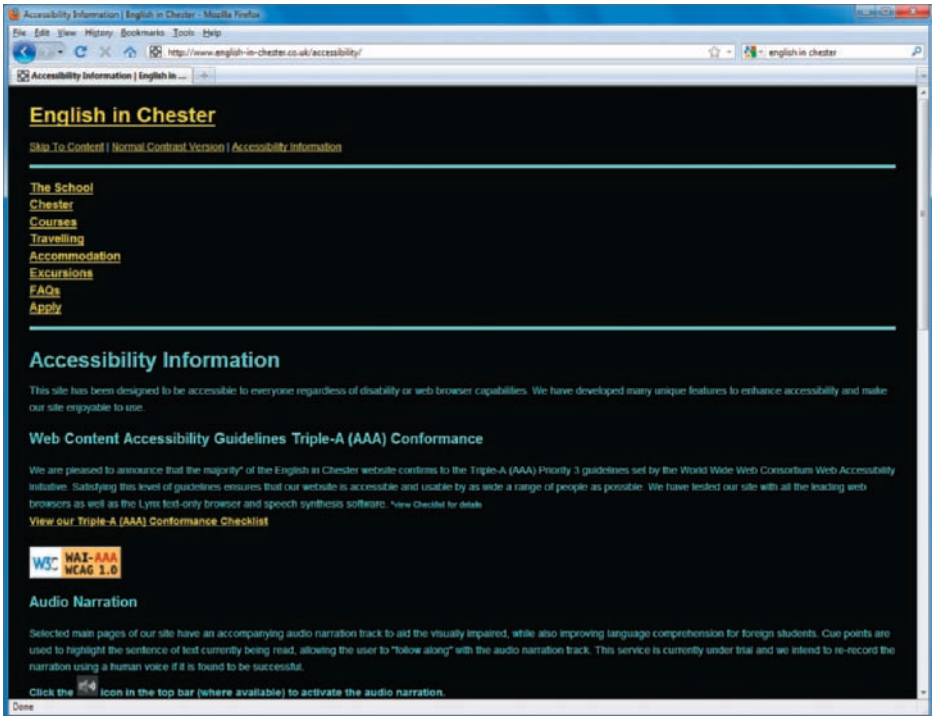


Figure 2-38 English in Chester Web site high-contrast version

To find out more about the W3C's Web Accessibility Initiative, go to *www.w3.org/WAI/*. The site includes many guidelines and standards to build accessible Web content and explains the release of WCAG 2.0 in detail. You can learn more about the adaptive devices and assistive technologies for accessible browsing at *www.w3.org/WAI/References/Browsing*. Finally, you can read the Section 508 guidelines at *www.section508.gov*.

Two current sets of accessibility guidelines are available to Web designers. The W3C's Web Accessibility Initiative publishes the WCAG 2.0. The U.S. government has its own set of guidelines, which are part of the Rehabilitation Act Amendments of 1998 called Section 508. The law requires federal agencies to provide information technology that is accessible to federal employees and citizens who have disabilities. Both sets of guidelines help you create more accessible Web content, so which should you use? If you are designing a Web site for the federal government, you must follow the 508 guidelines, but for general public Web sites, the W3C guidelines will suffice. The next section examines the WCAG 2.0 guidelines.

WCAG 2.0 Guidelines

WCAG 2.0 consists of four main guidelines, which state that Web content must be:

- *Perceivable*—Information and user interface components must be perceivable by users.
- *Operable*—User interface components must be operable by users.

- *Understandable*—Information about the user interface and its operation must be understandable by users.
- *Robust*—Content must be robust enough to be interpreted reliably by a wide variety of user agents, including assistive technologies.

To find a quick reference guide to the new WCAG 2.0, see *www.w3.org/WAI/WCAG20/quickref*.

A number of tips, summarized in the following sections, help define the goal of each guideline.

Perceivable Content

WCAG 2.0 includes the following tips to help you provide perceivable content in your Web pages:

- Provide text alternatives for any nontext content so that it can be changed into other forms people need, such as large print, Braille, speech, symbols, or simpler language.

  This guideline ensures that all nontext content is available as electronic text. For example, use the alt attribute to describe the function of each visual element. You will learn about using the alt attribute in Chapter 8.

- Provide synchronized alternatives for multimedia.

  This guideline ensures that users can access multimedia content. For example, provide captioning and transcripts of audio and descriptions of video as alternatives to the multimedia content.

- Create content that can be presented in different ways (for example, spoken aloud, in a simpler layout, and so on) without losing information or structure.

  This guideline ensures that information is available in a form that all users can perceive. For example, the best way to meet this guideline is to build well-structured content with separate presentation information (as you learned in Chapter 1). This practice ensures that the structure and relationship of the content does not change, no matter what assistive device or software the user chooses.

- Make it easier for people with disabilities to see and hear content, including separating foreground from background.

  This guideline ensures that the default presentation is as usable as possible to people with disabilities, making it easier for users to separate foreground information from the background. This basic tenet of Web design, often ignored, is to make sure that

information presented on top of a background contrasts sufficiently with the background. You will learn more about this principle in Chapter 8. This guideline also applies to spoken text, so that background sounds do not interfere with understanding the spoken text.

Operable Content

WCAG 2.0 includes the following tips to help you provide operable content in your Web pages:

- Make all functionality available from a keyboard.

  You should ensure that all actions that can be performed with a mouse or other input device can also be performed with a keyboard.

- Provide users who have disabilities enough time to read and use content.

  Many users with disabilities take longer to complete tasks than the average user. For example, some text scrolls animate text across the page faster than users with disabilities can read the text. This guideline ensures that these users can complete tasks within their comfort zone.

- Do not create content that is known to cause seizures.

  Some users with seizure disorders can have a seizure triggered by flashing visual content. The purpose of this guideline is to avoid content that flashes or blinks quickly enough to cause a seizure—for example, the deprecated <blink> element that causes text to flash on and off.

- Provide ways to help users with disabilities navigate, find content, and determine where they are.

  This guideline ensures that users can find the content they need and keep track of their location, as you read previously in the "Plan for Clear Presentation of Your Information" section in this chapter. This is a sound and essential design principle when planning for the general public as well as for users who have disabilities.

Understandable Content

WCAG 2.0 includes the following tips to help you provide understandable content in your Web pages:

- Make text content readable and understandable.

  This guideline ensures the greatest legibility and readability of text. Many characteristics of this guideline are simple, sound design techniques that you will read about in Chapter 7. These techniques include limiting the line length of text, providing appropriate white space, and avoiding large chunks of italic text.

- Make Web pages appear and operate in predictable ways.

  The purpose of this guideline is to help users with disabilities by presenting content and navigation options in a predictable order from Web page to Web page, as you learned in the "Create a Unified Site Design" section earlier in this chapter. Users who employ screen magnifiers see only part of the screen at one time. A consistent page layout makes it easier for them to find navigation bars and other repeated components. Placing page elements in the same relative order within a Web site allows users with reading disabilities to focus on the desired area of the screen, rather than spending additional time looking for what they want.

- Help users avoid and correct mistakes that do occur.

  This guideline ensures that users can detect when they have made an error—for example, when entering data into a form. Typical error indicators, such as color-coded text or an icon, may not be enough to alert the user. Additional indicators such as sound or highlighted text can aid the user in identifying incorrect entries.

Robust Content

WCAG 2.0 includes the following tips to help you provide robust content in your Web pages:

- Maximize compatibility with current and future user agents, including assistive technologies.

  This guideline ensures compatibility with current and future browsers, especially assistive technology browsers. To meet this guideline, use standards-compliant markup and validated code, as described in Chapter 1. Present content in standard ways that assistive technology software can recognize and with which it can interact.

You can verify that physically challenged people can access your Web pages easily by using *www.section508.info*.

Chapter Summary

Web sites have a wide variety of looks. It is easy to see why so many Web designers get caught up in the medium and forget their message. The lure of technology makes it easy to overlook that you are still trying to communicate with words and pictures, just as humans have for centuries. Adapting those elements to online display for effective communication is the challenge.

Plan a site that stands out and delivers its message. If you stick with the principles you learned in this chapter, you can present information that is both accessible and engaging.

- Craft an appropriate look and feel, and stick with it throughout your site. Test and revise your interface by paying close attention to the demands of online display.
- Make your design portable by testing it in a variety of browsers, operating systems, computing platforms, and connection speeds.
- Plan for easy access to your information. Provide logical navigation tools, and do not make users click through more than two or three pages before they reach the information they are seeking.
- Design a unified look for your site. Strive for smooth transitions from one page to the next. Create templates for your grid structure, and apply them consistently.
- Use active white space as an integral part of your design. Use text, color, and object placement to guide the user's eye.
- Know your audience, and design pages that suit their needs, interests, and viewing preferences.
- Leverage the power of hypertext linking. Provide enough links for users to create their own paths through your information.
- Design your text for online display, considering the differences between formatting for the screen and formatting for the printed page.
- Choose the suite of browsers and operating systems you will use to test your site. Although you will include the latest versions of Firefox, Safari, Opera, Internet Explorer, and Google Chrome, consider testing in older versions of each browser as well.

- Decide how browser specific your site will be. In most cases, your goal is to create a site that is widely accessible to multiple browsers. However, if you have a narrow audience or specific requirements, you may want to specify one browser as the primary method for viewing your site.
- Resolve to test your work continually as you build your site. Test with multiple browsers at different screen resolutions and at different connection speeds. If you can, view your site on multiple platforms such as Windows, Macintosh, and UNIX.
- Remember to build in accessibility from the beginning of your design efforts to make sure your content is available to everyone.

Key Terms

active white space—White space used deliberately as an integral part of your design that provides structure and separates content.

cache—The browser's temporary storage area for Web pages and images.

fixed-width page layout—A Web page layout that allows the designer to control the look of the Web pages as if it were a printed page, with consistent width and height.

flexible page layout—A Web page layout designed to adapt to different screen resolutions. Also called fluid layout.

grid—A layout device that organizes the Web page, providing visual consistency.

look and feel—The interface that the user must navigate on a Web site.

Media Queries—CSS statements that let you specify different style rules for different media destinations. For example, an HTML document could be displayed with different fonts for print or screen.

passive white space—The blank area that borders the screen or is the result of mismatched shapes in a layout.

screen resolution—The width and height of the computer screen in pixels.

Review Questions

1. Make a list of human factors to consider when building a Web site.
2. Make a list of technical factors to consider when building a Web site.
3. What design guidelines will you follow to ensure compatibility?
4. How does browser caching affect Web design?
5. How do multiple screen resolutions affect Web design?
6. Name three ways to create a unified look for your site.
7. Describe two methods of designing for multiple screen resolutions. Note the advantages and disadvantages of each method.
8. How does a grid layout enhance Web design?
9. Explain active versus passive white space.
10. List three ways to create a smooth transition between pages of a Web site.
11. List two benefits of consistently placing navigation tools.
12. Describe the difference between reading and scanning a page.
13. Describe three Web page viewing patterns
14. Name three ways to focus a user's attention.
15. Describe why using "Click Here" as link text is ineffective.
16. Describe the benefits of textual links.
17. Name three differences between paper-based and screen-based design.

18. Describe a good strategy to format text for online display.

19. Describe the four main guidelines in the WCAG 2.0 for designing accessible Web sites.

Hands-On Projects

1. Browse the Web for examples of Web sites that exhibit good Web design.
    a. Using a screen-capture program, such as the snipping tool in Windows 7.0, capture Web pages from the Web site that show two levels of information. For example, capture the main page of a Web site and a secondary page. Describe how the layouts for the two pages suit their information types and the needs of their users.
    b. Use the graphic tools in a word-processing program or presentation program to include screen callouts with the Web page images you capture. The callouts should indicate the unifying characteristics of the pages, such as shared colors, fonts, graphics, and page layout. (A callout is an arrow or line that connects to explanatory text. Many figures in this book have callouts, including Figure 2-2.) For example, Microsoft Word includes a callout tool in its collection of shapes.
    c. Indicate the areas of active white space and passive white space.
    d. Describe whether the design of the site is appropriate for the content.

2. Browse the Web for examples of poor Web design on mainstream Web sites.
    a. Using a screen-capture program, capture Web pages from the Web site that show two levels of information. For example, capture the main page of a Web site and a secondary page.
    b. Indicate with screen callouts the jarring or distracting inconsistencies of the site, such as abrupt changes in any design elements, including theme and layout.
    c. List detailed recommendations for improving the site design.

3. Write a short essay critiquing a Web site's design. Describe the structural layout of the site and determine whether information is presented clearly and is easily accessible.

4. Browse the Web for sites that use unique navigation methods. Write a short essay describing why the method is or is not successful, or prepare a short slide presentation that you can present to your class.

5. Find a Web site that you think needs improvements in its design.
   a. Print two pages from the site.
   b. Make copies of the originals, and set the originals aside.
   c. Using scissors, cut out the main elements of each page. Rearrange the elements and paste them in a design you believe improves the site.
   d. Compare and contrast the original with your improved design.

6. Test cross-browser compatibility.
   a. Make sure you have recent versions of at least two current browsers installed on your computer. (See "Hands-on Project 1" in Chapter 1 if you need to download and install a browser.)
   b. Browse a variety of Web sites. Make sure to view various pages of the sites in the different browsers.
   c. Write a detailed description of how the various sites appear in the browsers you have chosen. Look for text, layout, and graphic inconsistencies.

7. Test accessibility software. Download a trial version of one of the following screen-reading programs. Navigate to different Web sites, and use the tool to read the site. Close your eyes while listening. Write a short essay describing your experience.
   - Jaws at *www.freedomscientific.com/downloads/jaws/jaws-downloads.asp*
   - Window Eyes at *www.gwmicro.com/Window-Eyes/Demo*

Individual Case Project

Visualize the page design for your site by sketching a number of page layouts for different information levels of the site. Figure 2-39 shows a sample page sketch. For example, sketch the main page, a secondary page, and a content page. You do not have to be concerned with the exact look of the elements, but be prepared to indicate the main components of the pages, such as headings, navigation cues, link sets, text areas, and so on.

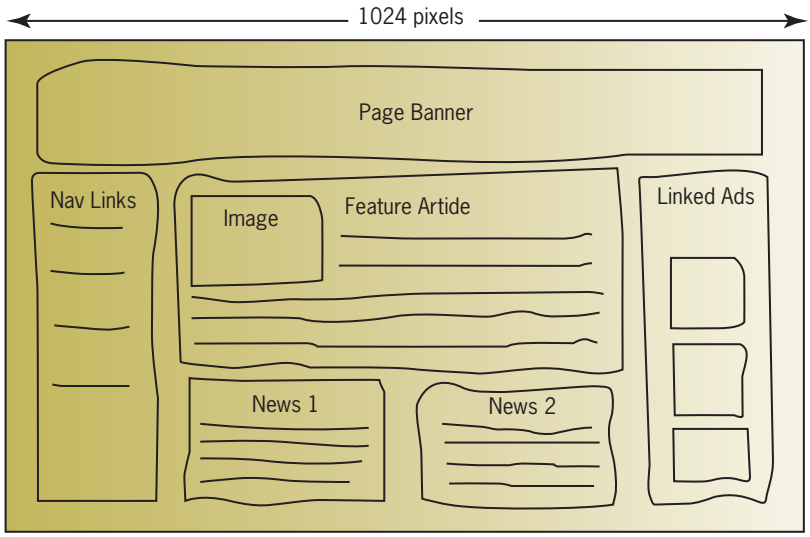


Figure 2-39 Sample page layout sketch

Start to organize your site. Create a visual diagram that indicates the main page, section pages, content pages, and so on. Indicate the links between the pages. Indicate whether you will provide alternate navigation choices such as a table of contents and site map.

Team Case Project

Work individually to create your view of the page designs for your site. Sketch a number of page layouts for different information levels of the site. For example, sketch the main page, a secondary page, and a content page, similar to the example shown in Figure 2-39. You do not have to be concerned with the exact look of the elements, but be prepared to indicate the main components

of the pages, such as headings, navigation cues, link sets, text areas, and so on.

Next, meet and work as a team to create the team's page layouts by combining the best ideas from the individual page designs. Organize your site. Create visual diagrams that indicate the main page, section pages, content pages, and so on. Indicate the links between the pages. Indicate whether you will provide alternate navigation choices such as a table of contents and site map.

CHAPTER 3

Site Planning

When you complete this chapter, you will be able to:

- Understand the Web site development process
- Create a site specification
- Identify the content goal
- Analyze your audience
- Build a Web site development team
- Create conventions for filenames and URLs
- Set a directory structure
- Create a site storyboard
- Publish your Web site
- Test your Web site

A good Web site design requires a detailed initial planning phase. Before starting to code your site, pick up a pencil and paper and sketch out your site design. Creating the stylistic conventions and conceptual structure of your site beforehand saves time during development. Whether you are creating a single personal Web site or working on a professional Web development team, you save time and improve quality by thoroughly planning the site design and development process before you start creating it. This chapter walks you through planning and building a framework for your site, resulting in more efficient development when you build your Web site.

Understanding the Web Site Development Process

What are your objectives for building a Web site? You may want to gain visibility, provide a service, sell a product, create a community, attract new customers, or disseminate information. Although the content may vary, a good project outcome requires a sound development process to ensure that you have valid and achievable goals for your site.

A good project plan encompasses all stages of the project and is accessible to everyone involved. The complexity and depth of each stage of the planning process can vary based on the complexity of the site. In commercial Web site development, a project manager controls and disseminates the project plan using tools such as Microsoft Project to create charts and track all phases of production.

You can choose from many types of models of the Web site development process and use one as a framework for planning your site development. Figure 3-1 shows a typical high-level project plan. The stages in this illustration would match scheduled calendar dates that are milestones in the project plan. The life-cycle of the project is the complete project plan from inception to completion. Stages can overlap at different points in the process. As the project evolves, design changes must decrease to ensure a successful implementation, meaning that the beginning of the project is where most design experimentation should occur. As the project progresses, the design must become stable for successful completion. Major design changes cannot be introduced late in the project without significant rework and risk of extending the project schedule.

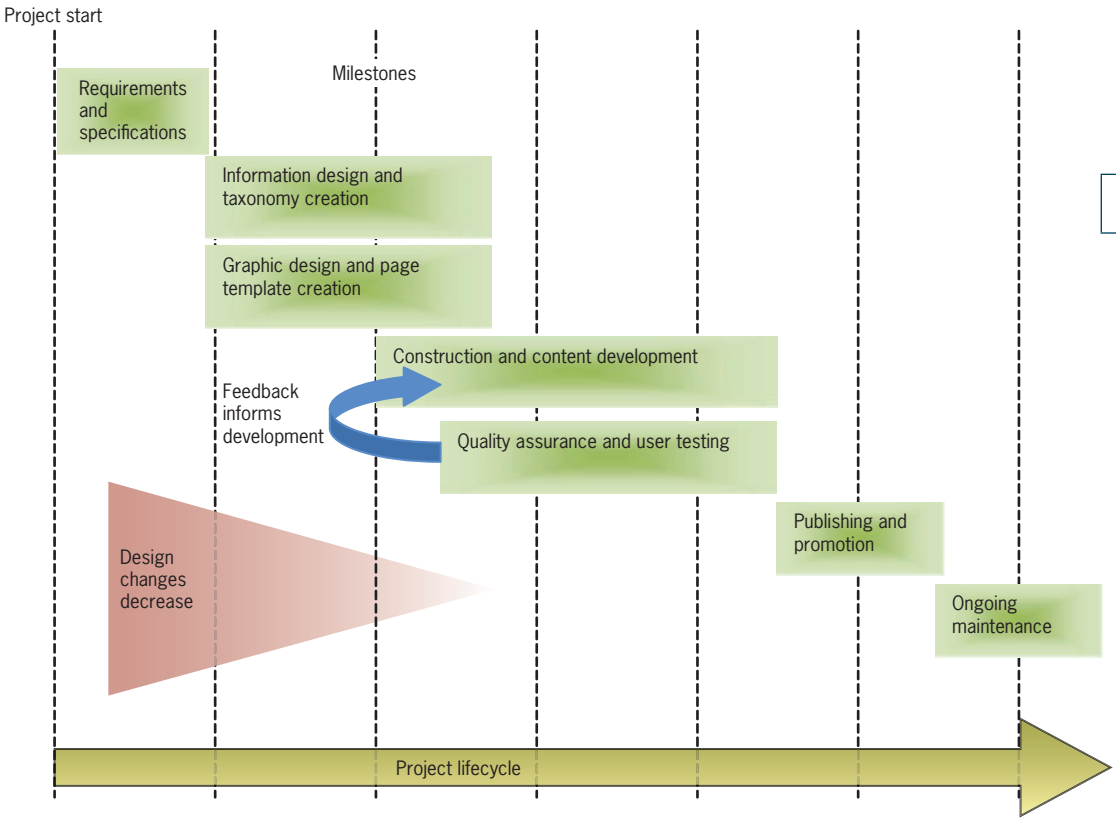


Figure 3-1 Web development project lifecycle

Requirements and Specification

In this stage of a Web development project, the client presents the requirements for the Web site. **Requirements** are the list of customer needs, such as search capability, tabbed menu navigation, specific color and branding requirements, or anything else that will create the desired outcome for the site. The Web project team must analyze these requirements for viability and then break them down into tasks. This is a good time to assess the talents of the team members to make sure all requirements can be met. For example, if a particular programming language is required, then a team member must have those skills to meet the requirement. The Web project team works with the client during this stage to analyze and define the audience. After analyzing and defining requirements and determining the user profile, the team prepares a project

specification that contains the design requirements, page layout sketches, audience definition, and technical requirements. As a Web designer, the specification will guide you to the next stages of the project.

Information Design and Taxonomy Creation

In this stage, user analysis guides the designers as they prepare and test different organizations of the site content. The goal of this stage is to structure the site content in a way that is the most meaningful and easiest to navigate by the intended user. During this stage, the taxonomy of the site information is developed. A **taxonomy** is a classification and naming of content in a hierarchical structure. The taxonomy of the site directly translates to the navigation through the top-level content topics down to individual pieces of information that the user is looking for. The taxonomy is often reflected in topic section names and in the navigation and menu system of the site.

Graphic Design and Page Template Creation

At the same time designers are testing the organization of the site information, they prepare design sketches and page mockups to represent page layouts that will be used in the site. All page layouts start with a mockup that is usually just a sketch of the desired design. Designers can submit the page layout mockups to the Web site stakeholders for discussion and critique. Generally, you create a mockup for each page layout in the Web site. A sketch of a proposed design, as shown in Figure 3-2, indicates the general layout of a Web site home page.

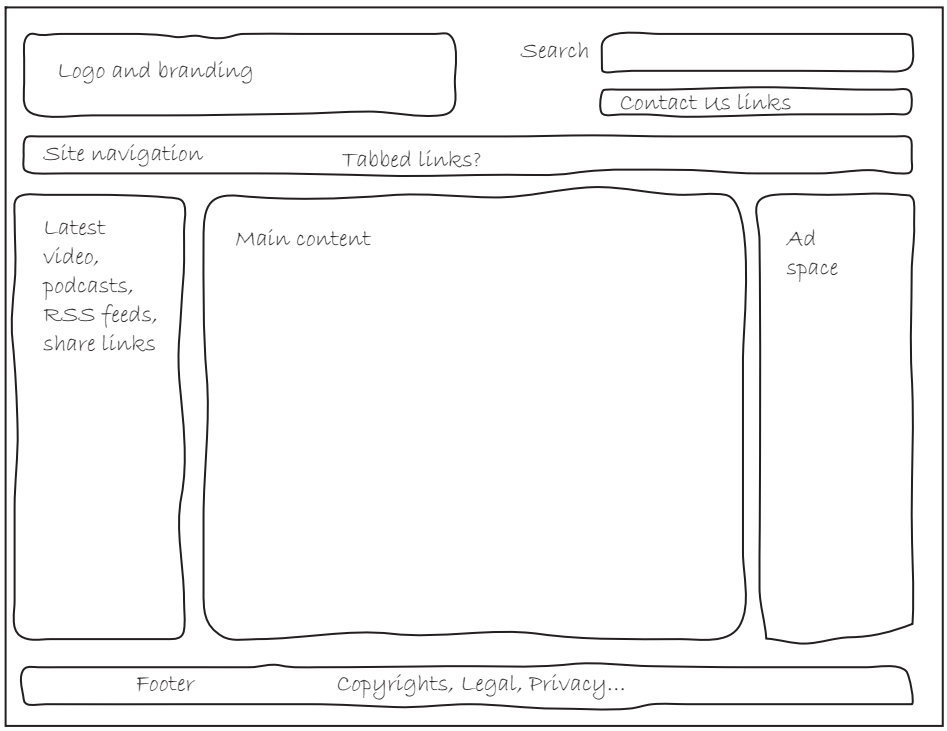


Figure 3-2 Web page mockup

These mockups can be easily edited and changed based on feedback and input from the design team. As the design becomes more stable, the mockup can evolve to a more refined state, often called a **wireframe**. Wireframes show a more complete version of the page designs, often including navigation elements, search functions, advertising space, and other similar elements. The wireframes offer stakeholders a more complete view of what the final design will look like. Designers also use wireframes to gain insight and reactions from content developers and software engineers and to test design changes. Remember that it is always easier to make changes to a page mockup or wireframe than it is to a Web site once you have started coding it.

Figure 3-3 shows a wireframe that builds on the page design articulated in the mockup sketch in Figure 3-2.

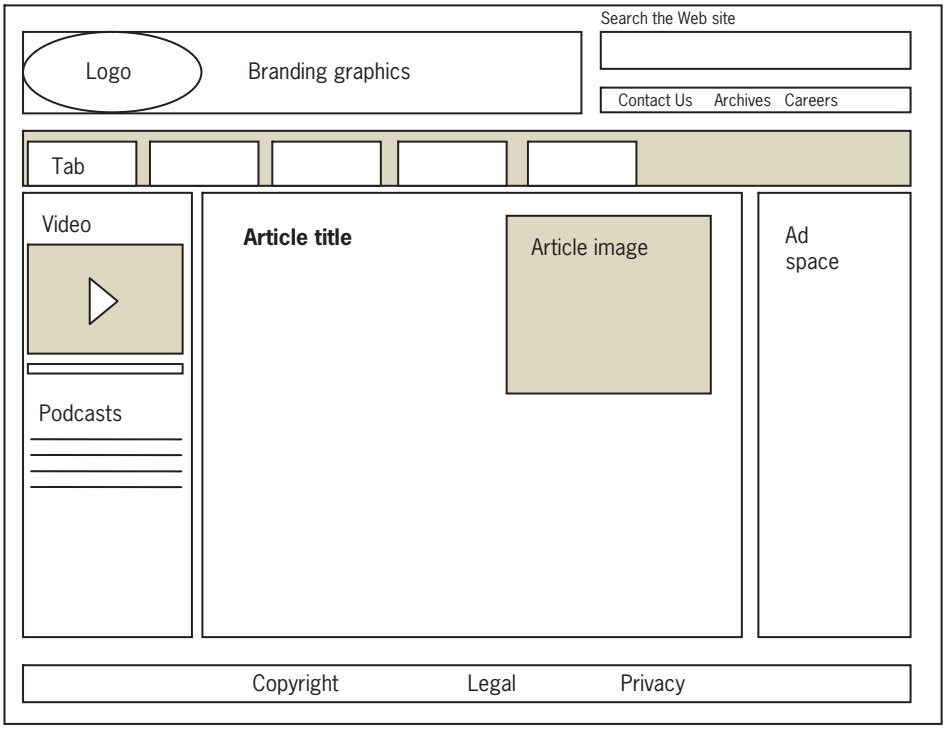


Figure 3-3 Sample wireframe for page layout

You can find freeware wireframe software tools at the following Web sites:

- Pencil (*http://pencil.evolus.vn/en-US/Home.aspx*)
- Gliffy (*www.gliffy.com/wireframe-software*)

Construction and Content Development

When the design stage is mostly complete and page designs are stable, the construction stage can begin. This stage encompasses all of the technical development of the site, including page coding and validation, application development, and content preparation. Some testing occurs during this stage of loading content in page templates and evaluating the performance of applications or multimedia.

Quality Assurance and User Testing

As the site construction nears completion, the quality assurance and user testing phase validates the design of the site. The development team performs various tests for cross-browser compatibility, accessibility to all users, and connectivity at different bandwidths. They test links and all user interfaces, data forms, and multimedia technologies. Usability testing ensures that users can access content, navigate through the site, and understand the taxonomy.

Publishing and Promotion

During this stage, the site is published to the Web or the organization intranet, and the client begins to publicize and promote the site. This includes making the Web site address available in all collateral media, such as print and broadcast; advertising on other Web sites; registering with search engines; distributing press releases; and starting publicity campaigns.

Ongoing Maintenance

This stage begins at the moment the site goes live, as Web content must be updated and kept fresh to remain vital. New sections of content may be added that restart certain phases of the project lifecycle, such as new audience definitions, designs for new content areas, or the design and development of new interface elements or interactive features.

Creating a Site Specification

Start your planning by creating a **site specification**; this is the design document for your site. If you completed the Individual or Team Case Project at the end of Chapter 1, you created a basic draft of a project proposal. You can use some of that information in your site specification. After you read this chapter, you will be able to answer additional questions about your site. You can return to the site specification as you build your site to help maintain your focus. If you are providing a Web site design to a client, the site specification is the first document the client sees that establishes the basic site design. Answer the following questions in your site specification:

You can read a sample site specification in Chapter 12.

- Who is the client for the Web site? This is not the user, but the person who has employed you to build a Web site. Are you creating a personal site (in which case, you are the client), or are you part of a design team working independently or for a corporation or nonprofit organization? What do you or the client's company or organization hope to gain from creating and maintaining a Web site?
- Can you write a mission statement that succinctly states the site's focus and goals?

- What are the requirements for the Web site? Look to your client for their list of wants. Explicitly state the required functions that you want the site to contain.
- Are the requirements feasible? Do you or your team have the necessary technical and editorial skills to build the site that the client requires?
- How will you judge the success of the site? What are the factors you can use to assess the effectiveness of the site?
- Who is the target audience? What are some common characteristics? How can you find out more about your target audience?
- What are the limiting technical factors affecting your site?
- What is the budget? What is the schedule or target milestone dates? Are the dates realistic and achievable?
- Is this a new site or an upgrade to an existing site? If this is a site upgrade project, what can be learned from the first version of the site?

Identifying the Content Goal

Consider carefully what type of site you are building. What you and the design team want the Web site to accomplish and what your users want from your site may differ. For example, designers and other stakeholders are often more concerned with the look than the feel. Don't let this deter you from advocating for the user. Your users probably care more about how quickly they can find information. Adopt your user's perspective, and let your user analysis guide your content decision. Think about the type of content the Web site will provide and what will give it the greatest value to your user. Look at your search and navigation capabilities carefully to ensure ease and direct access. Look to the Web for examples of how best to present, organize, and focus your content. The following types of Web sites demonstrate ways to focus your content.

- *Billboard*—These sites establish a Web presence for a business or commercial venture. In many cases, they are informational and offer limited content, acting as an online business card or brochure rather than offering Web-based interaction. Many smaller businesses build this type of site first and then expand as necessary, adding functions when needed.

- *Publishing*—Every major newspaper and periodical publishes both to print media and to the Web. These Web sites are some of the most ambitious in breadth and depth of content, often containing multiple levels of information with many page designs. Many publishing sites use content management systems (CMS) to dynamically create Web pages, drawing content from the same databases that produce their paper-based versions. This allows their authors to write the article once, but have it published to multiple destinations, such as the daily newspaper and the Web site.
- *Portal*—Portals act as gateways to the Web and offer an array of services including searching, e-mail, shopping, news, and organized links to Web resources. Many major search engines have been converted into portals to attract more users. These sites are often heavy with advertising content, which is their main source of revenue.
- *Special interest, public interest, and nonprofit organization*—These sites include news and current information for volunteers, devotees, novices, a specific audience, or the general public. Public-service Web sites contain links, information, downloadable files, addresses, and telephone numbers that can help you solve a problem or find more resources. Nonprofit organizations can state their manifestos, seek volunteers, and foster grassroots virtual communities.
- *Blog*—Short for "Weblog," a blog is a personal Web page that reflects the personality and interests of the author. No matter what your interest, a community of *bloggers* (blog authors) on the Web is devoted to it. Many blogs are personal diaries or commentaries on life. Most blogs are published with tools that can archive content, provide comment threads that allow visitors to comment on blog posts, and offer built-in page designs. WordPress (*www.wordpress.com*) is a popular example of this type of software. There are literally millions of blogs on the Web. You can find listings at *www.blogcatalog.com* and *www.blogarama.com*.
- *Social networking*—Sites such as MySpace, Facebook, Bebo, and LinkedIn are the most visible of these types of virtual communities, which allow users to post profiles and connect with friends and family. Users can share comments and exchange messages with their friends, play games, take surveys, and post photos, videos, and links to interesting sites. The popularity of social networking sites has risen sharply in the past few years. They are often the reason many people use the Internet.

- *Wikis*—A wiki is a type of online database that accepts contributions from multiple authors. Wikis are collaborative Web sites that allow contributors to use wiki software to easily edit information and create linked Web pages. Wikis allow any user to comment on or change another user's entries. Wikipedia is the most visible example of a wiki, but they are appropriate anywhere collaboration and sharing of content is needed, and are used in both academic and commercial environments.
- *RSS (Real Simple Syndication)*—This is a service provided by Web sites. More technically, RSS is a format for Web feeds that automatically update users who have subscribed for this service. RSS feeds usually contain headlines or summaries of content, which are read using a software tool called an RSS Reader. RSS Readers are built into the major browsers. With RSS feeds, any author updates to a Web site are automatically transmitted to subscribers.
- *Virtual gallery*—The Web is a great place to show off samples of all types of art and design. Photographers and artists can display samples of their work; musicians and bands can post audio files of their songs; writers can offer sections of text or complete manuscripts. However, keep in mind that any copyrighted material you display on a Web site can be downloaded to a user's machine without your permission. As a solution to this problem, software companies such as Digimarc (*www.digimarc.com*) offer digital watermarking technology that lets artists embed digital copyright information in their electronic files as a deterrent to piracy of proprietary content. This information cannot be seen or altered by the user.
- *E-commerce, catalog, and online shopping*—The Web as shopping medium continues to expand as more users improve their Internet access and learn to trust the security of online commerce. Web commerce competes successfully with traditional retailing, offering many advantages over mail-order shopping, such as letting the customer know immediately whether an item is in stock. Other types of commerce on the Web include stock trading, airline ticketing, online banking, auctions, and more. Many software vendors offer turnkey systems that can be integrated with existing databases to speed the development of a commerce site. A good electronic commerce (e-commerce) site provides users with quick access to the item they want, shopping carts and wish lists to store their choices, detailed product descriptions, and easy, secure ordering.

- *Product support*—The Web is a boon to consumers who need help with a product. Manufacturers can disseminate information, upgrades, troubleshooting advice, documentation, and online tutorials through their Web sites. Companies that provide good product support information on the Web find that the volume of telephone-based customer support calls decreases. Software companies especially benefit from the Web; users can download patches and upgrades and use trial versions of software before they buy.
- *Intranet and extranet*—Private Web sites are often hosted on a company intranet or extranet. An **intranet** is a smaller, limited version of the Internet on a company's private local area network (LAN), accessible only to those who are authorized to use their network. Many companies have telecommuting employees who need access to company policies, documentation, parts lists, pricing information, and other materials. These employees can be reached via an **extranet**, which is a part of the private intranet extended outside the organization via the Internet. Web sites on an intranet or extranet are typically developed by companies for use by employees, customers, and suppliers. Many organizations mandate a particular browser for employee use, making the Web designer's job a little easier, because they only have to code and test for one browser.

Analyzing Your Audience

If possible, analyze your audience and produce an **audience definition**, a profile of your average user. If you are building a new site, work from your market research, look at sites with content similar to yours, and try to characterize your average user. If you have an existing user base, contact your typical users and try to answer the following questions:

- What do users want when they come to your site? Can they easily find desired content?
- How can you initially attract visitors and entice them to return?
- What type of computer and connection speed does your typical visitor have?

Though your users may fit no common profile, you can gather information about them in a few ways. One way is to include

an online feedback form in your site. Figure 3-4 shows a sample online survey from the State of Maine Web site (*www.maine.gov/portal/survey.html).*

The survey asks users about their experiences visiting the Web site. It uses both scaled (or rated) and open-ended questions to elicit a variety of responses from the user concerning the visual and information design of the site.

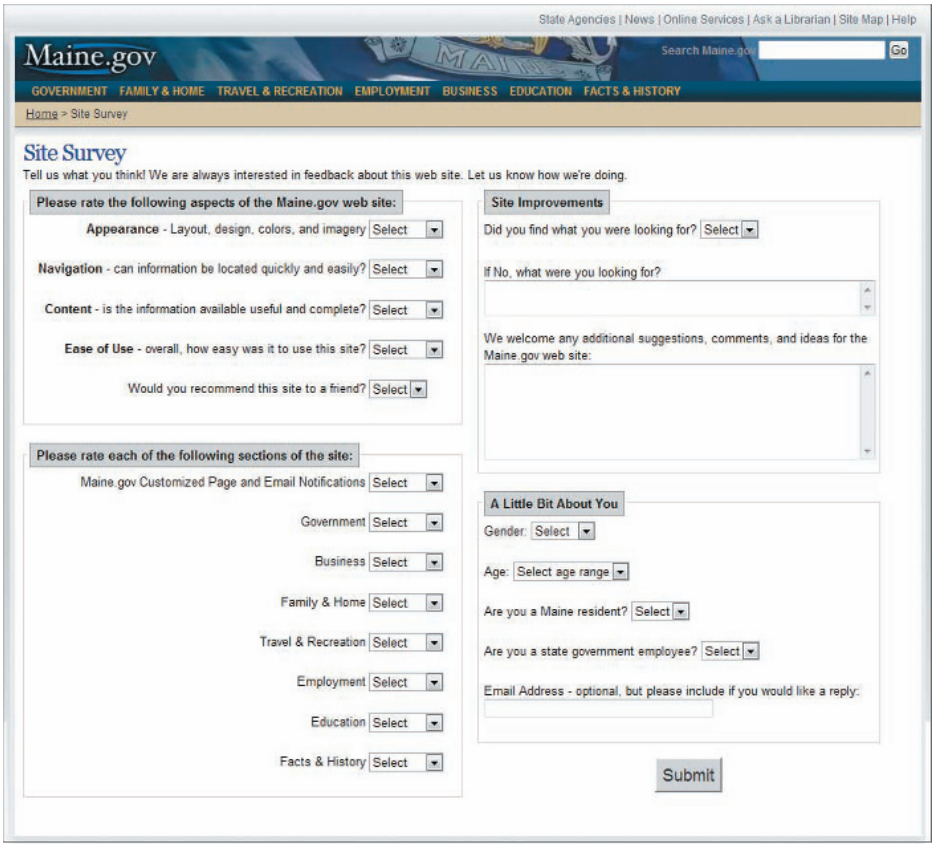


Figure 3-4 Sample user feedback form

If you cannot survey your users, or if you feel you are not getting good survey results, try to adopt a typical user's perspective as you define your audience. Here are some questions to consider:

- Who are the typical members of your audience? Are they male or female? Do they have accessibility issues? What is their level of education? What is their reading and vocabulary level? What is their level of technical aptitude?

- Why do people come to your site? Do they want information? Do they want to download files? Are they looking for links to other Web sites?
- Do you have a captive audience, such as a base of loyal customers that want up-to-date information? Are you designing for an intranet, where users are employees of an organization?
- If users are unfamiliar with the site, will they know what you offer?
- How often will users return to your site? Why would they come back?
- What computing platform do your users have? What is their typical connection speed? What type of browser do they use? If you are on an intranet, does it use standards for browsers, connection, and screen resolution?
- Whose skills do you need to build the site? Who will create the graphics, code the pages, and write the text? Do you have the talent and economic resources that you need? Will the results meet the expectations of your users?

Refine your content and presentation even after your site is built and running. Continue soliciting user feedback to keep your site focused and the content fresh.

You can set up a free Web-based survey with SurveyMonkey (*www.surveymonkey.com*).

Using Web Analytics

Web analytics are statistics that are gathered by Web servers and then analyzed. Web servers track and record various usage and traffic statistics in files called server logs. Reporting tools can aggregate and analyze the data in these files to help you learn about your users and their activity on your Web site. The data gathered by the Web server contains relevant details about your visitors, including how they came to your site, whether by bookmark, search engine, or by clicking a link on another Web site. If visitors used a search engine, you can see what term they entered to find your site. You can see what pages are most visited, and even see an approximate view of where your users are geographically located, based on their identifying Internet address.

Figure 3-5 shows a sample of statistics from W3 Counter (*www.w3counter.com*), a free Web analytics tool that you can add to any Web site. This figure shows two important Web log statistics: page views and unique visitors. A **page view** is the number of times a page is viewed by a user, which could be multiple times by the same user. The unique visitor statistic tracks the number of unique visitors to a site, no matter how many times they access the same page. So the page views tell you which pages are the most popular with your users, while the unique visits tell you how many different users are visiting your site.

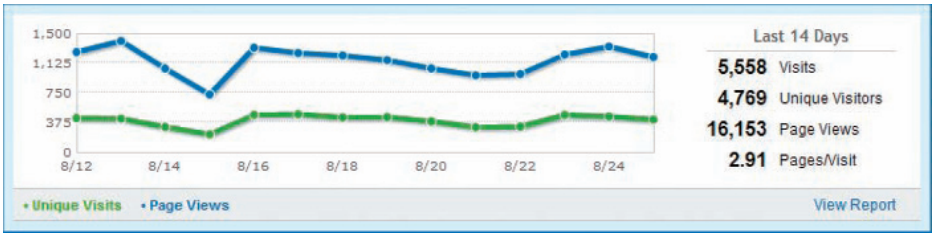


Figure 3-5 Sample Web analytics results

In addition to viewing how many times users accessed your pages and visited your site, you can also see where your users originated from geographically, as shown in Figure 3-6.

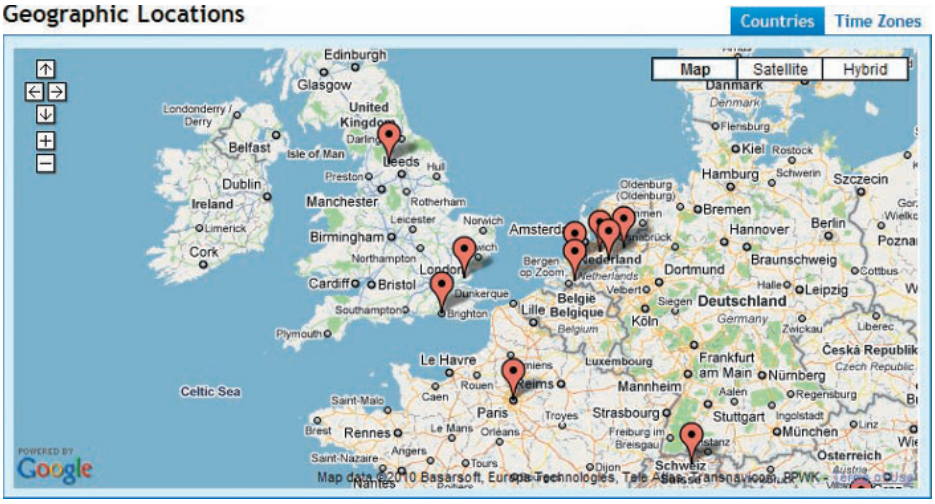


Figure 3-6 Sample geographical results

Web analytics are an important method of determining details about your users and how they interact with and traverse the content areas of your site. Use Web analytics as a guide to enhancing your content and making important pages accessible, but remember that they are not flawless. Many factors can affect the results you will receive. Statistics are just one method of determining user preferences.

Identifying Technology Issues and Accessibility Constraints

Make your best effort to identify any technological limitations or advantages that members of your audience share. As you read in Chapter 2, you have to make assumptions about the user's browser, connection speed, operating system, and screen resolution. Users in developed countries have greater access to the latest hardware technologies and consistent high-speed Internet connections. Many other users throughout the world may be using older equipment. Think about where your users are located and what their technology level might be. Test in different environments and with different technologies to make sure these users can view your site.

You also need to consider the physical capabilities of users that are visiting your site. How do they interact with your content? Are they older users with sight or dexterity issues? Will they need to magnify your pages to read your text? Many types of physical limitations can affect the way people interact with a computer.

When planning a Web site, you can identify accessibility constraints in the following ways:

- Review the WCAG 2.0 and Section 508 accessibility guidelines as needed.
- If you are building a new Web site, plan for accessibility.
- If you have an existing Web site, assess the current accessibility of your content.
- Review case study examples of real-life accessibility implementations.
- Discuss accessibility solutions, authoring tools, and evaluation tools with other Web professionals.

Study your Web statistics and user feedback. If you think your user is the average person browsing the Web, you may have to adopt settings that represent the lowest common denominator to satisfy the widest audience. If you find that your users are savvy about technology and use the latest computer hardware and software, higher resolution and connection speeds may apply. If you are designing an intranet site, you may have the luxury of knowing your users' exact operating systems and browser versions. Whatever the particulars, make sure to design at an appropriate level, or you risk losing visitors.

Identifying Software Tools

Determining the software requirements for your Web site is important during the planning process. Try to choose software that matches the complexity and needs of your site so that you do not end up with a tool that is either underequipped or overspecialized. Simple Web sites, including many student sites, can be built with one of the many shareware and freeware tools (see Table 3-1) that are available on the Web. As your site and skills grow, you might choose to move up to more robust tools such as Adobe Creative Suite or individual Adobe tools such as Dreamweaver (*www.adobe.com*). Microsoft offers Expression Web as its Web site design tool. These tools offer complete coding, design, and site management capabilities. You may also need graphics tools (discussed in Chapter 8), database software, and online credit and shopping programs, based on the skills and talents of the members of your Web site team, as described in the next section.

One popular type of software is **shareware**, programs that you can download and use for a trial period. After the trial period, users can register the software for a relatively small fee compared to commercially produced software. Another type of software already mentioned in this chapter is **freeware**, which is available free of charge or with an optional donation fee if you want to contribute to support the software developers' efforts. Table 3-1 lists freeware HTML development tools.

Development Tool	Platform	URL
Amaya	Windows, Macintosh, Linux	*www.w3.org/Amaya*
BBEdit	Macintosh	*www.barebones.com*
Bluefish	Windows, Macintosh, Linux	*bluefish.openoffice.nl*
Firebug Firefox add-on	Windows, Macintosh, Linux	*addons.mozilla.org/en-US/firefox/addon/1843*
Kompozer	Windows, Macintosh, Linux	*www.kompozer.net*
SeaMonkey	Windows, Macintosh, Linux	*www.seamonkey-project.org*
Trellian	Windows	*www.trellian.com/webpage*

Table 3-1 List of Freeware Web Site Development Tools

Building a Web Site Development Team

Although one person can maintain small Web sites, larger sites require groups of people filling a variety of roles. Of course, the line between these roles can be blurred, and many aspects of site design require collaboration to solve a problem. The following are examples of the types of talent necessary to build a larger, well-conceived site.

- *Project management*—The project management team is responsible for planning, scheduling, and integrating the many tasks that it takes to create a Web site. They create the milestones for deliverables and balance the staffing resources to keep the project on schedule and within budget. The project manager coordinates communication among team members and keeps the focus on the deliverables promised to the client.
- *HTML developers*—These are the people responsible for creating the HTML code, conforming to standards, validating code, troubleshooting the site, and testing the site across different operating systems and Web browsers.
- *Designers*—Designers are the graphic artists responsible for the look of the site. They use graphic design software such as Adobe Photoshop or Adobe Fireworks. Designers are responsible for the wireframes, page template design, navigation icons, color scheme, and logos. If your site uses photographic content, the designers are called upon to prepare the photos for online display. They might also create animations and interactive content using Adobe Flash.

- *Writers and information designers*—Writers prepare content for online display, including taxonomies, hypertext linking conventions, and navigation paths. In addition, many writers are responsible for creating a site style guide and defining typographic conventions, as well as consistency, grammar, spelling, and tone. They also work closely with the designers to develop page templates and interactive content.
- *Application developers*—Developers write the software programs and scripts you need to build interaction into your site. They may write a variety of applications in different programming languages for user interaction or write back-end applications that interact with a database.
- *Database administrators*—The people who are responsible for maintaining the databases play an important role in commercial Web sites. Databases store all the information for customer transactions and e-commerce. Database administrators, application developers, and HTML developers work together when designing front-end forms used to collect data from the user. Database administrators are also responsible for data security backup and data recovery.
- *Server administrators*—Get to know and appreciate the technical people who run your Web server. They take care of the sticky technical issues such as firewalls, ports, internal security, file administration, and backup procedures. Consult with them to determine your Web site's default filename and directory structure. They also can manage the server logs that contribute to your Web analytics reporting to determine how many visitors your site is attracting, where the visitors are coming from, and what pages they like best.

Creating Conventions for Filenames and URLs

Before you sit down at the keyboard, plan the filename conventions for your site. Find out from your system administrator what type of operating system your Web server uses. Typically you develop your Web site locally on a PC or Macintosh and upload the files to the Web server as the last step in the publishing process. If the Web server runs a different operating system from your local development system, any filename or directory structure inconsistencies encountered in transferring your files to the server may break local URL links.

Naming Files

A filename's maximum length, valid characters, punctuation, and sensitivity to uppercase and lowercase letters vary among operating systems, as described in Table 3-2. Note that the **ISO 9660 Standard** is the base file-naming convention designed to work across all operating systems.

Operating System and File System	Filename Conventions
ISO 9660 Standard	Maximum of eight letters followed by a period and a three-letter extension; allowed characters are letters, numbers, and the underscore (_)
Newer PCs: Windows 7, Windows Vista, Windows XP (NTFS), Windows 2000 (NTFS), Microsoft Windows/NT (NTFS)	Maximum of 255 letters, all characters allowed except \ / * " < > | : ?
Older PCs: Windows 98 (FAT32), Windows 95 (VFAT), DOS, and Windows 3.× (FAT file system)	The same as ISO 9660 but with the following additional characters allowed: $ % ‘ ` - @ ^ ! & [ ] ( ) # This format is also compatible with newer PC operating systems
Newer Macintosh: O/S 8.1 to OS X	Maximum of 255 characters, all characters allowed except the colon (:)
Older Macintosh: Operating systems released before O/S 8.1	Maximum of 31 letters, all characters allowed except the colon (:) This format is also compatible with newer Macintosh operating systems
UNIX	Maximum of 255 letters, all characters allowed except the forward slash (/) and spaces

Table 3-2 File Naming Conventions

Case Sensitivity

If you have an image file named Picture.gif, for example, and you reference that file as , the image is displayed properly on a Macintosh or Windows machine. On a UNIX server, however, the image does not load properly because UNIX is case sensitive; *Picture.gif* and *picture.gif* are recognized as two different files. It is best to use lowercase letters for all filenames, including filenames in your HTML code.

Character Exceptions

As shown in Table 3-2, it is best when naming your files to leave out special characters such as <, >, /, \, &, *, and blank spaces to ensure cross-platform compatibility. Some special characters that may be valid on one operating system will not work on another.

File Extensions

You must use the correct file extensions to identify your file to the browser. HTML text files created in HTML-editing programs commonly end in .htm or .html unless they are generated dynamically by an application. In this case, they may have extensions such as .asp, .php, or others. You also must use the correct filename extensions to identify image file formats. For example, Joint Photographic Experts Group (JPEG) files must end in .jpg or .jpeg; Graphics Interchange Format (GIF) files must end in .gif; and Portable Network Graphic (PNG) files must end in .png.

Choosing the Correct File-Naming Conventions

It is best to set conventions for your filenames right from the beginning of the Web development process. Create a list of conventions and refer to it frequently. Here are some guidelines to remember:

- Don't use spaces in your filenames; use underscores instead. Instead of *about web design.html*, use *about_web_design.html*.
- Avoid all special characters. Stick to letters, numbers, dashes, and underscores.
- Use all lowercase letters for your filenames.

Default Main Page Name

Every Web site has a default main page that appears when the browser requests the main URL of the site, such as www.google.com rather than a specific file. In this instance, the Web server must decide which file to provide, which is usually the home page of the site. The default is generally *index.html*, but others may apply based on site particulars. Always check with your system administrator to verify the correct main page filename.

Using Complete or Partial URLs

Although you may know that URLs are the addresses you type into your browser to access a site, you may not realize that there are two types of URLs: complete and partial.

Complete URLs

A **Uniform Resource Locator (URL)** is the unique address of a file's location on the World Wide Web. A **complete URL** includes the protocol the browser uses, the server or domain name, the path, and the filename. Figure 3-7 shows an example of a complete URL.

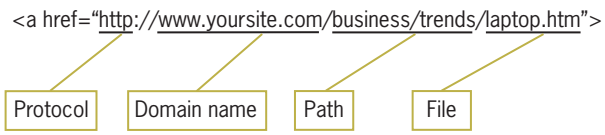


Figure 3-7 Parts of a complete URL

In this example, *http* is the protocol, and *www.yoursite.com* is the domain name. The path shows that the destination file, *laptop.htm,* resides in the *business/trends* folder. Use complete URLs in your HTML code when linking to another Web site.

Partial URLs

Use a partial URL when you are linking to a file that resides on your own computer or server. **Partial URLs** omit the protocol and domain or server name, and specify the path to the file on the same server. Files that reside in the same directory (or folder) need no path information other than the filename. The following code shows an example of a partial URL.

```
<a href="laptop.htm">link text</a>
```

Setting a Directory Structure

You will probably build your Web site on a computer that is different from the computer that hosts your site. Keep this in mind when you are designing the directory and file structure. All of the files for your Web site will need to be transferred from your computer to the Web server that will be hosting your site. Because your files will be transferred to another computer, any URLs you specify to link to other pages in your site must include paths that

are transferable. This is why you should never specify an absolute path in your partial URLs. An absolute path points to the computer's root directory, indicated by a leading (forward) slash in the file path:

```
/graphics/logo.gif
```

If you include the root directory in your partial URLs, you are basing your file structure on your development machine's file system. If the files are moved to another machine, the path to your files will not apply, and your site will include links that do not work because the browser cannot find the files.

To avoid this problem, use relative paths. Relative paths tell the browser where a file is located relative to the document the browser currently is viewing. Because relative paths are not based on the root directory, they are transferable to other computers.

Using a Single Folder Structure

One easy way to ensure that all your path names are correct is to keep all of your HTML and image files in the same directory. Because all files are kept together, the only information you need to put in the src or href attribute is the filename itself. In Figure 3-8, User2 has a simplified directory structure. To reference the file logo.gif, User2 adds the following code in one of the HTML files:

```
<img src="logo.gif" />
```

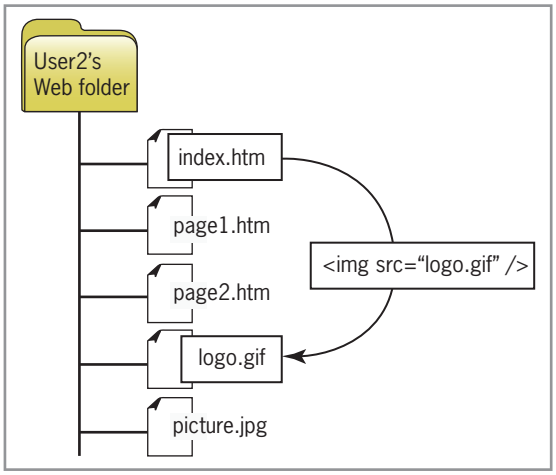


Figure 3-8 Simplified single folder file structure

Using a Hierarchical Folder Structure

The simple directory structure shown in the preceding example is fine for a small Web site, but as your site grows you may want to segregate different types of content into separate folders for ease of maintenance. Take a look at the relative file structure for User2's Web site as depicted in Figure 3-9. Notice that User2's Web folder contains three HTML files and one subfolder named images, which contains the graphics and pictures for the Web site.

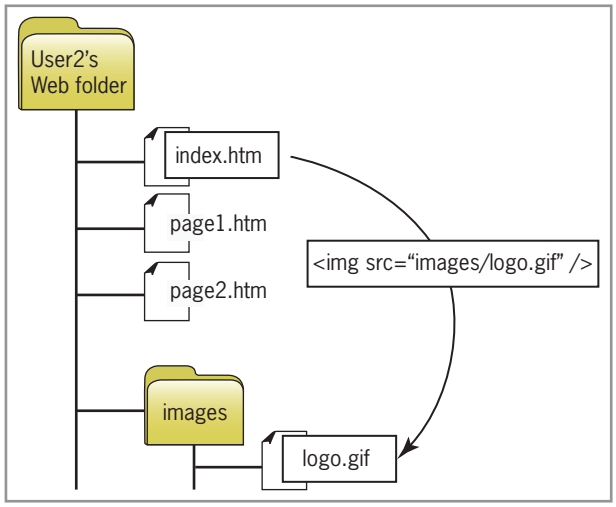


Figure 3-9 Basic hierarchical folder structure

To include the image file logo.gif in index.htm, User2 adds the following code to index.htm:

```
<img src="images/logo.gif" />
```

The path in the src value tells the browser to look down one level in the directory structure for the images folder and find the file logo.gif. The path to the file is relative to the file the browser is viewing. This type of relative file structure can be moved to different machines; the relationship between the files does not change, because everything is relative within the Web folder.

User2's Web site may need a more segregated directory structure, as shown in Figure 3-10. In this example, common files such as the index (the home page) and site map reside in the top-level folder. Multiple subfolders contain chapter and image content. Two linking examples are illustrated in this figure:

- *Example 1*—To build a link from page1.htm (in the chapter1 folder) to index.htm, use ../ in the path statement to indicate that the file resides one level higher in the directory structure, as shown in the following code:

```
<a href="../index.htm">Home</a>
```

- *Example 2*—To include the image file logo.gif in page1.htm, use ../ to indicate that the file resides in the images folder, which is one level higher in the directory structure, as shown in the following code:

```
<img src="../images/logo.gif" />
```

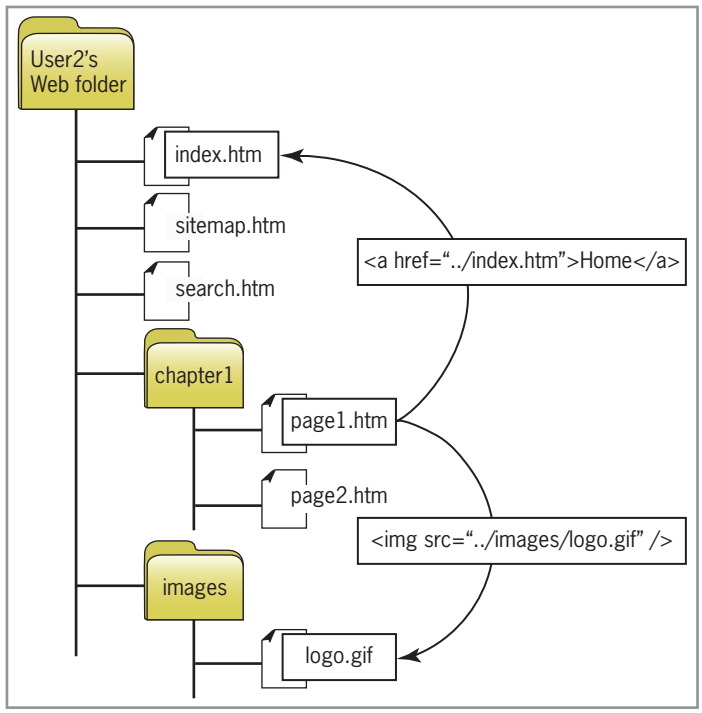


Figure 3-10 More segregated hierarchical folder structure

Creating a Site Storyboard

Plan your site by creating a storyboard flowchart that shows the structure, logic, and taxonomy behind the content presentation and navigation choices you offer. You can sketch your site with paper and pencil or create it using flowchart diagramming software. Sometimes it is helpful to use sticky notes or cards to plan the structure visually. This method lets you easily move pages from one section or level to another. Whichever method you choose, this preliminary planning step is one of the most important in

planning your site. You can move pages and whole sections of content freely, plan navigation paths, and visualize the entire site. This is the stage at which to experiment and refine your designs. Once you have started coding the site, it is much more difficult and time consuming to go back and make major changes. Remember to adhere to the file-naming conventions for each of your pages.

Organizing the Information Structure

Think about your users' information needs and how they can best access the content of your site. How should your information design map look? Review the sample structures provided in this section, and judge how well they fit your information. Your design may incorporate several structures, or you may have to adapt the structures to your content. Each sample structure is a template; you may have more or fewer pages, sections, topics, or links. You may choose to use bidirectional links where only single-direction links are indicated. Use these examples as starting points and design from there.

Linear Structure

The linear information structure, illustrated in Figure 3-11, guides the user along a straightforward path. This structure lends itself to booklike presentations; once into the content, users can navigate backward or forward. Each page can contain a link back to the main page if desired. Pages may also contain links to a related subtopic. If the users jump to the subtopic page, they only can return to the page that contains the subtopic link. This structured navigation returns them to the same point in the content path.

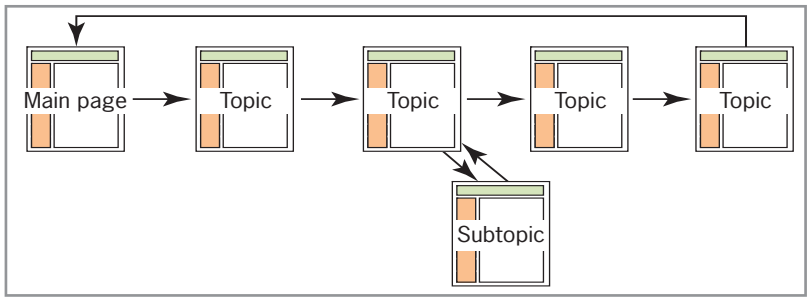


Figure 3-11 Linear information structure

Tutorial Structure

The tutorial structure illustrated in Figure 3-12 is perfect for computer-based training content such as lessons, tutorials, or

task-oriented procedures. The tutorial structure builds on the simple linear structure in Figure 3-11. The user navigates the concept, lesson, and review pages in order. Because the lessons use hypertext, users can leave the lesson structure and return at any time. They also can choose the order of lessons and start anywhere they want. Notice that the table of contents, index, and site map pages are linked to—and from—all pages in the course. Within each lesson users can navigate as necessary to familiarize themselves with the content before they review. This structure can be adapted to fit content needs; for example, the group of pages in the illustration could be one section of a larger training course.

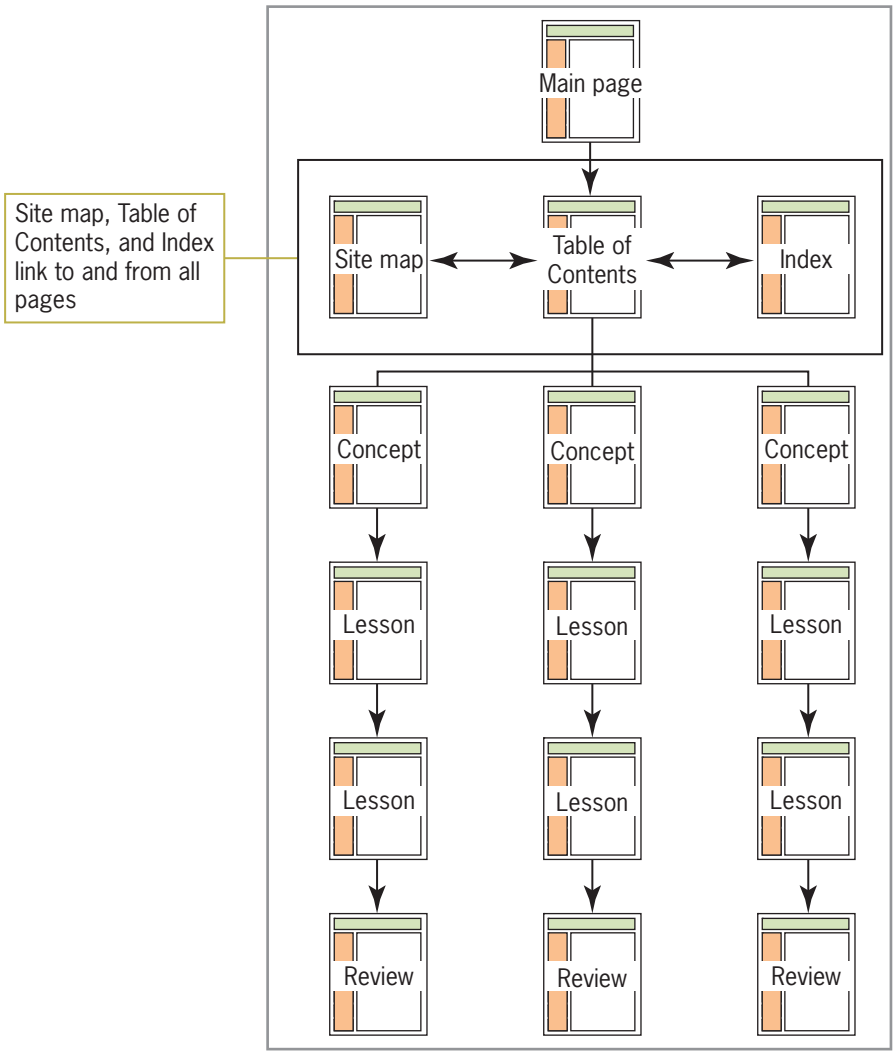


Figure 3-12 Tutorial structure

Web Structure

Many smaller Web sites follow the Web-type content structure illustrated in Figure 3-13, which is nonlinear, allowing the user to jump freely to any page from any other page. If you choose to use this type of content structure, make sure that each page includes clear location information and a standardized navigation bar that not only tells users where they are, but where they can go.

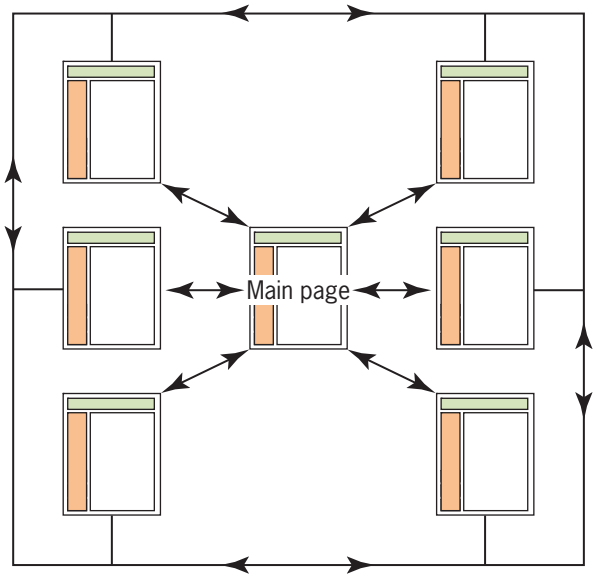


Figure 3-13 Web structure

Hierarchical Structure

The hierarchical structure illustrated in Figure 3-14 is probably the most common information design. It lends itself to larger content collections because the section pages break up and organize the content at different levels throughout the site. Navigation is primarily linear within the content sections. Users can scan the content on the section page and then choose the content page of their choice. When they finish reading the content, they can return to the section page. The site map allows users to navigate freely throughout the site. A navigation bar on each page lets the user jump to any section page, the main page, and the site map.

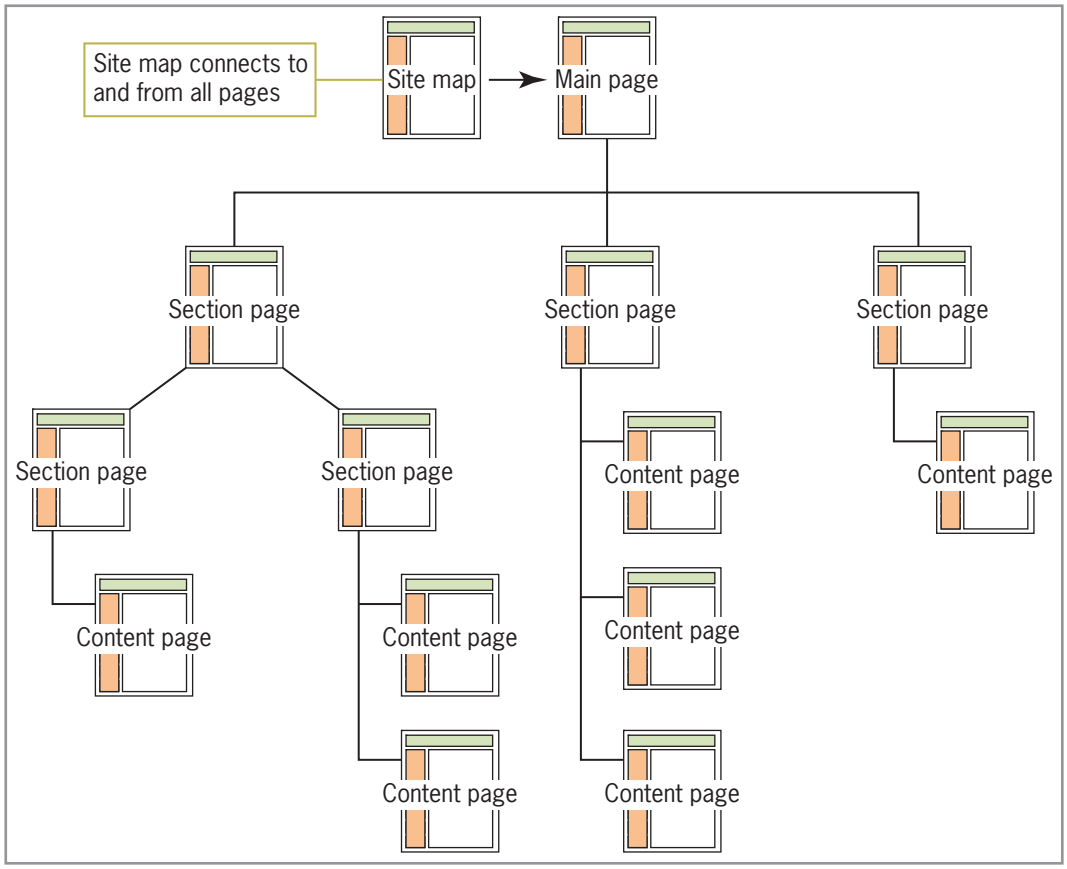


Figure 3-14 Hierarchical structure

Cluster Structure

The cluster structure illustrated in Figure 3-15 is similar to the hierarchical structure, except that every topic area is an island of information, with all pages in each cluster linked to each other. This structure encourages exploration within a topic area, allowing the user to navigate freely through the content. All pages contain a navigation bar with links to the section pages, main page, and site map.

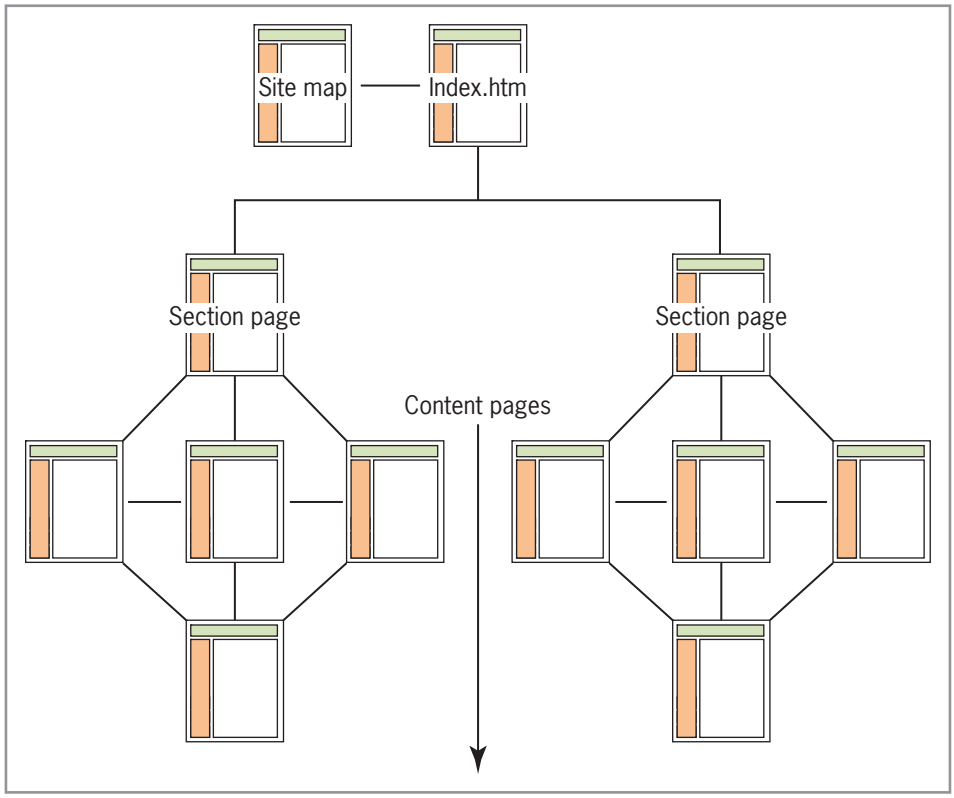


Figure 3-15 Cluster structure

Catalog Structure

The catalog structure illustrated in Figure 3-16 is ideally suited to electronic shopping. The user can browse or search for items and view specific information about each product on the item pages. Users can add items to their shopping cart as they shop. When they are finished, they can review the items in their shopping cart and then proceed to checkout, where they can enter credit card information and finalize the order.

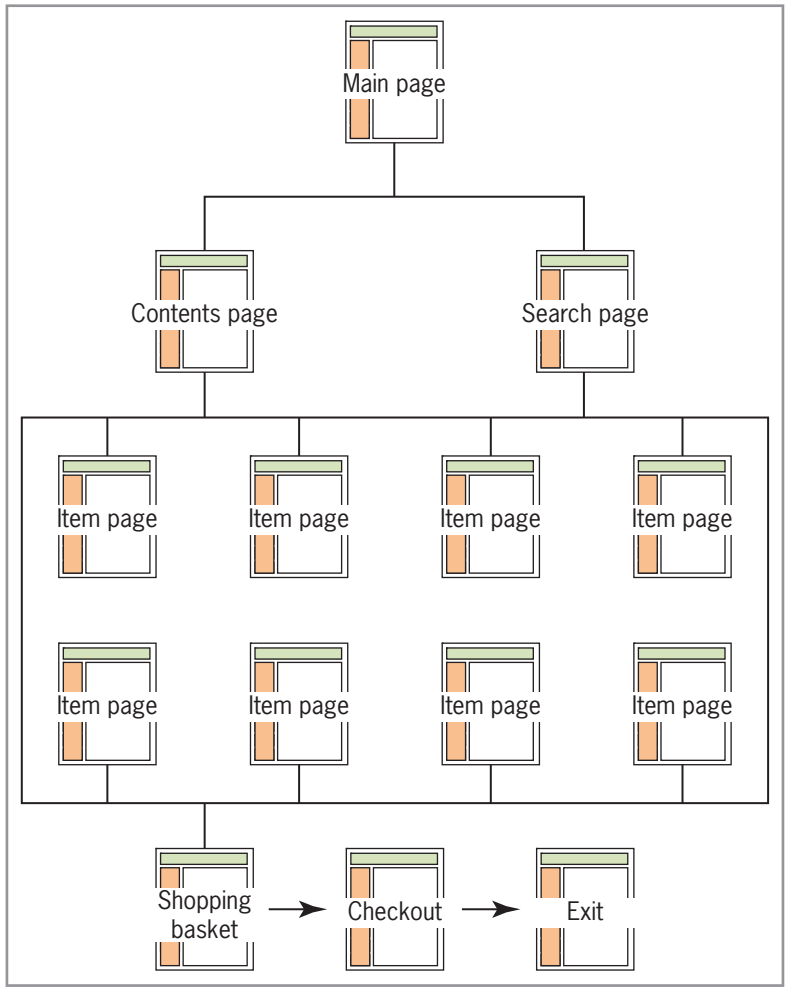


Figure 3-16 Catalog structure

This type of Web site requires back-end data transaction processing to handle the shopping cart tally, process credit card information, and generate an order for the warehouse. Businesses that want to set up an e-commerce site can purchase ready-made commerce software packages or develop their own from scratch.

Publishing Your Web Site

To make your Web site live, you transfer your Web site files to a **Web server**, a computer connected to the Internet and running server software. The software lets the computer use the Hypertext Transfer Protocol (HTTP) to serve HTML files to Web browser clients. Unless your company or organization has a Web server and hosts its own content, you must use the services of a Web hosting provider. After you choose a server to host your files, you need to select file transfer software and upload the Web site files from your development machine to the Web server.

Choosing a Web Hosting Service Provider

One of the most important choices you will make is your Web hosting service. This is the company that hosts your Web pages on a Web server, making them available to anyone who knows your URL. Most Web hosting companies offer hosting services for both personal and business use. The Web host provides you with Internet access, e-mail accounts, and space for a personal or business Web site. If you are building a Web site for business use, your Web host can register a personalized domain name for your Web site.

Small Web sites (around 15–20 pages of content) do not need much more than 1 or 2 MB of server space to hold all of the HTML pages and graphics. Your Web hosting package should provide at least 10 MB of space so your Web page has room to grow. Many personal Web sites can be hosted on the free server space that comes with many cable and digital subscriber line (DSL) modem connection packages. Check with your service provider to see if this feature is available.

Larger or more complex sites need more server space, especially if you have downloadable files, archives, lots of graphic content, or databases. If you are building a business Web site, seek out larger hosting services that are more appropriate for hosting a complex commercial site.

Shopping for a Web hosting service can be a confusing experience, as no two are exactly alike. Do some research and learn about offerings from different vendors. The following sections discuss the features you should seek in a hosting service.

Some Web hosting services offer proprietary design tools and templates to assist you in building a Web site. It's best to avoid these types of tools as they tend to tie you to one vendor and make it difficult to switch hosting services and post your Web site elsewhere.

DSL and Cable Access

Most Internet users now have access to high-speed, broadband connection services through a DSL or cable modem. To take advantage of DSL or cable access to the Web, you need a network card for your computer and a DSL or cable modem. **Internet service providers (ISPs)** usually supply a modem with their service. Check to make sure that the monthly fee does not include the equipment costs for the modem. Because DSL and cable are "always-on" connections, they introduce an increased security risk that can make your network vulnerable to hackers. If your provider does not offer network security, you must purchase a network security device, known as a gateway router, to protect your computer with a security firewall. The router allows multiple computers in your home or business to share the high-speed Internet connection, while the firewall software blocks intruders from accessing your network.

Accessible Technical Support

Technical support is not a feature—it is an absolute necessity. Make sure that your Web hosting service has competent, accessible customer service. When you are checking into Web hosting services, call and talk with the companies' customer service representatives. Tell them how experienced you are with computers, and let them know what you hope to accomplish (such as the type of Web site you want to build). Note how long you are on hold when waiting to speak with customer service. Make sure that you are comfortable with the level of service you receive on these initial inquiries.

E-Mail Addresses

All Web hosting accounts come with variable number of e-mail addresses that you can assign to yourself and anyone else you want to have an e-mail address that uses your domain name. If you are part of a group, multiple mailbox accounts let each person receive his or her own e-mail. You can also set up a "catch-all" e-mail address that will accept any e-mail sent to your Web site regardless of whether it is addressed to you.

SQL Database Support

If you are planning on any type of electronic commerce or customized data presentation, you need database support. Databases that understand **Structured Query Language (SQL)**, a programming language that lets you select information from a database, are the most common and powerful type of database.

Secure Sockets Layer (SSL) Support

The **Secure Sockets Layer (SSL)** is an Internet communications protocol that allows encrypted transmission of data between the user and the server. SSL is necessary if you are planning to set up an electronic commerce site or transmit other sensitive data. Encrypting the data ensures the information cannot be read if the transmission is intercepted.

Registering a Domain Name

A domain name is an alias that points to your actual location on the Web server, as shown in Figure 3-17. User2 has purchased the domain name *www.mysite.com*. The actual path to User2's content is hidden, and the visitor to the site sees only the domain name.

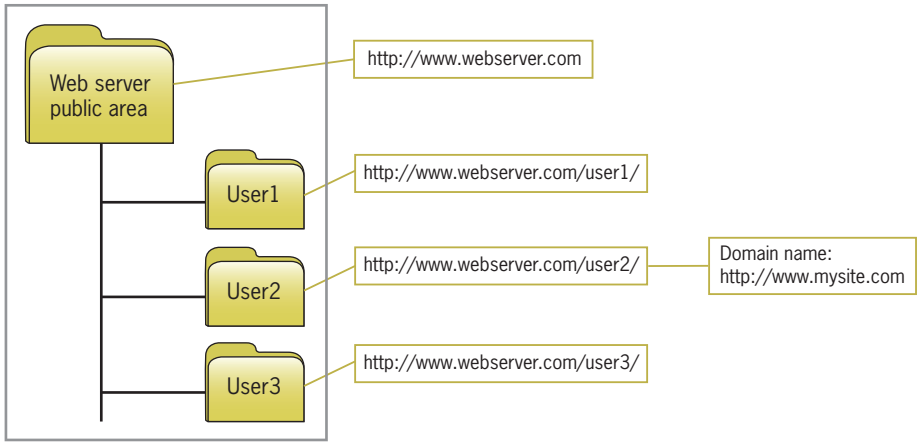


Figure 3-17 Domain name hides the actual path

Domain names are managed by the Internet Corporation for Assigned Names and Numbers (ICANN). ICANN has agreements with a number of vendors to provide domain name registration services. You can purchase a domain name through a vendor, and they will register it with ICANN. Current popular domain name registration services include Go Daddy (*www.godaddy.com*), Register.com (*www.register.com*), and Dotster (*www.dotster.com*).

Web Hosting Service Comparison Checklist

Use the following checklist when you compare Web hosting services:

- Is the Web host local or national?
- What are the details of the different hosting packages? How much server space comes with each? What are the limits (if any) on uploads and downloads?
- Are there bandwidth limits for the number of visitors your site receives per month?
- Does the Web host offer technical support? When is support staff available?
- How many e-mail addresses do you get with an account?
- Does the Web host provide software, such as a **File Transfer Protocol (FTP)** client to transfer files over the Internet?
- Does the Web host support the latest connection technologies?
- Does the Web host offer enhanced services, such as SQL database support, SSL, a scripting language environment, and support for streaming audio and video?

Always keep a backup of your Web site files in case you have any problems during FTP transmissions, or if you accidentally delete or overwrite existing files. Of course, if you accidentally delete or overwrite files on your local computer, you can always use your Web site files as a backup.

Uploading Files with the File Transfer Protocol

To publish your pages on the Web, you must send your HTML code, image, and other files to the Web server. To do this, you need FTP software, often called an FTP client. Some HTML-authoring software, such as Microsoft Expression Web and Adobe Dreamweaver, include built-in software packages that let you upload files to your Web server. You also can choose from many shareware and freeware FTP programs to upload your files. Table 3-3 lists some popular FTP clients available as freeware and shareware. Visit your favorite shareware site, such as Shareware.com, and search for FTP clients.

FTP Client	Web Site
Cute FTP—Shareware	*www.cuteftp.com*
Filezilla—Freeware	*http://filezilla-project.org*
FireFTP (Firefox plug-in)—Freeware	*http://fireftp.mozdev.org*
Fugu (Macintosh)	*http://rsug.itd.umich.edu/software/fugu*

Table 3-3 Freeware and Shareware FTP Clients

When you have decided which FTP software to use, verify your correct FTP address from your Web host. You also need your account name and password, which in most cases automatically points your FTP program to the proper directory on the server.

To upload your files, start your FTP program and connect to your Web server using the FTP information provided by your service provider. Your password allows you write access to your directory on the Web server, which means you can copy files to and from the directory. Once the FTP client has connected to the Web server, you have the option of choosing the files you want to transfer. The FTP client usually displays directories on both the local and remote computers. Figure 3-18 shows the FireFTP plug-in for Firefox, which lets you transfer files using your browser. You can see that the files on the local computer and Web server match, as files on the Web server are duplicates of the files maintained on the local computer.

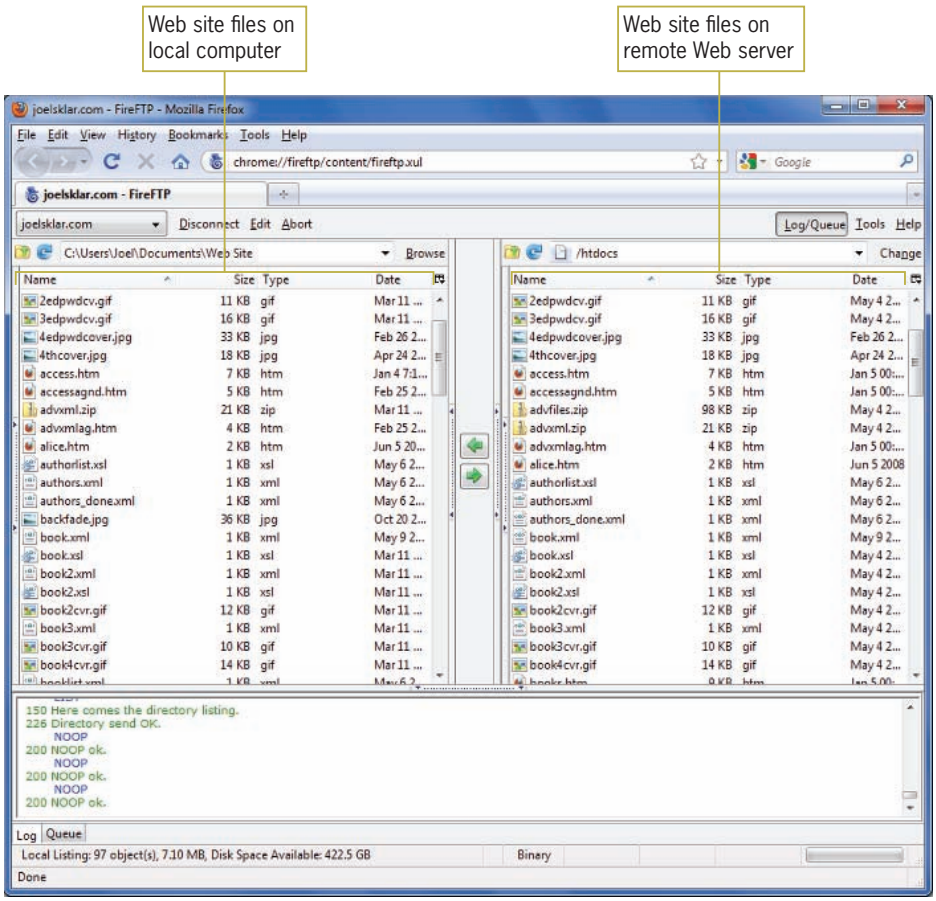


Figure 3-18 Typical FTP software

As discussed earlier in this chapter, make sure that you maintain the exact directory structure on the Web server that you used on your development computer to ensure that all relative file paths are correct.

Select the files that you want to upload in your local directory listing and transfer them to the Web server. You also can transfer files from the Web server to your computer. The first time you go live with your Web site, you must transfer all the files. Later you upload only the files that you have updated. After the files reach the Web server, they are available for access immediately on the Web.

After you find a Web hosting service and publish your Web site to the World Wide Web, it is time to test your Web site in the real-life Internet environment.

Testing Your Web Site

Even though you performed tests throughout the development of your Web site, you need to continue testing after you post your files live on the Web. If possible, load your files to the Web server and test them before making your URL available for users to access the Web site. If you have enough server space, you may want to establish a testing area on the Web site. You can do this by creating a subdirectory in your public HTML directory. Do not publicize the URL so that your testing area can remain private.

Testing Considerations

Always test in as many different environments as possible. Remember to test for the following Web design variables:

- *Multiple browsers*—Test your site using as many browsers as you can to make sure your work is portable and is displayed consistently.
- *Multiple operating systems*—If you can, test your site from different operating systems. If you have a PC as a development machine, use a Macintosh for testing, and vice versa. You even can run different versions of UNIX on a PC, if necessary. Because computer chip development moves at a lightning pace, machines become outdated quickly. You can find discounted and used machines that often are Internet-capable as long as they have a good Internet connection. Because you won't use these machines to *develop* Web sites (only to *view* them), you do not need the latest or most powerful hardware.

- *Connection speeds*—Do not rely on the same connection speed when testing your Web site, especially if you work in a corporate environment where the connection to the Internet usually is faster than the average user's. Go to a friend's house, library, or Internet café and access your Web site from there. Test for download times at different connection speeds.
- *Display types*—Test at different screen resolutions and monitor types to make sure your colors are displayed consistently.
- *Link testing*—Use a link validation tool to ensure that all of your links connect to a live page. Link validation tools are built into many HTML editors and are available as stand-alone tools. Many Web sites also offer validation, including the W3C's link validator at *validator.w3.org/checklink*. Any pages that link outside of your Web site need to be tested on a regular basis to make sure that the destination site has not moved, shut down, or posted content different from what you expect.

Usability Testing

Usability testing can be as simple as asking a few colleagues to look at your Web site, or as complex as conducting extensive formalized testing. Some companies invest in special user testing labs with videotaping and one-way mirrors to record user behavior, or software that can track users' mouse movements and eye coordination as they look at their Web site. Even if you do not need this level of sophisticated testing, you should perform some type of user assessment of your work. The goal of user testing is to determine whether your Web site is easy to navigate with easy access to content. Following are some points to consider when planning for user testing of your site.

Vary Your Subjects

Draw your test subjects from a variety of backgrounds, if possible. Gather test subjects who are representative of your target audience. Find users with varying computing skills and familiarity with the information. Avoid using friends as test users, as they may only compliment your work. You might choose to let users look at the Web site on their own time, but you can learn a lot by watching users interact with your Web site. Make sure to let them navigate and use the Web site without any outside help from you. Just stand back and watch.

Formalize Your Testing

Formalize your testing by creating replicable methods of testing your Web site. Prepare a series of questions that users have to answer after viewing the Web site. Give them a specific task to complete or have them find a particular piece of information. Let them rate the ease of completing such tasks. Compare the results from different users to find any problem areas in navigation. Administer the same testing methods to a variety of users, and watch for trends and consistencies. This lets you compare results or focus on a particular feature of the Web site.

Develop a Feedback Form

Develop a feedback form that users can fill out after they have tested the Web site. Include a set of criteria, and let them rate the Web site on a progressive scale, or ask them a series of open-ended questions. You also may want to provide the feedback form online, letting users offer feedback directly from the Web site. Here are some sample questions you might ask:

- Did you find the information you needed?
- Was it easy or difficult to access the information you needed?
- Did you find the Web site visually attractive?
- Did you find the content easy to read?
- Did you find the Web site easy to navigate?
- Did you think the information was presented correctly?
- Did the information have enough depth?
- What area of the Web site did you like the best? Why?
- What area of the Web site did you like the least? Why?
- Would you recommend the Web site to others?

Chapter Summary

A successful Web site is the result of careful planning. The steps you take before you actually start coding the site save you time, energy, and expenses in the long run. Remember these guidelines for successful planning:

- Become familiar with the stages in the Web site development cycle.
- Start with pencil and paper; your ideas are less restricted and you can easily revise and recast without recoding. Follow a development process to ensure fewer revisions and design reworking.
- Write a site specification document. You will find it invaluable as a reference while building your site.
- Identify the content goal by adopting your users' perspective and learning what they expect from your site.
- Analyze your audience and create an audience profile. Focus your site on the users' needs, and continue to meet those needs by adapting the site based on user feedback.
- An effective site is most commonly the result of a team effort. Leverage different skill sets and experience to build a Web site development team.
- Plan for successful implementation of your site by creating portable filename conventions. Build a relative file structure that can be transferred to your Web server without a hitch.
- Select a basic information structure for your site, and then manually diagram it, customizing it to the needs of your site.
- Shop carefully and compare features when you are looking for an ISP or Web host. Consider the future disk space and technology needs of your content.
- Download and learn to use an FTP client for use in the often-repeated task of transferring files to your Web site.
- After your Web site is live, test it against the basic Web variables of browser, operating system, display resolution, and connection speed.
- Test your Web site with a variety of users. Listen carefully to their feedback to identify trouble spots in your information design.
- Plan for the maintenance, upkeep, and redesign of your Web site. Keep your content up to date. Let users know when you have made updates to the Web site.

Key Terms

audience definition—A profile of your average user.

complete URL—An address of documents and other resources on the Web that includes the protocol the browser uses to access the file, service, domain name, the relative path, and the filename.

extranet—A private part of a company's intranet that uses the Internet to securely share part of an organization's information.

File Transfer Protocol (FTP)—A standard communications protocol for transferring files over the Internet.

freeware—Programs available free of charge or with an optional donation fee.

Internet service provider (ISP)—A company that provides Internet access and Web site hosting services to individuals and organizations.

intranet—A private collection of networks contained within an organization. Intranet users gain access to the Internet through a firewall that prevents unauthorized users from getting in to the intranet.

ISO 9660 Standard—A file system standard published by the International Organization for Standardization (ISO) that supports computer operating systems such as Windows, classic Mac OS, and UNIX-like systems, to exchange data.

page view—The number of times a Web page is viewed by a user.

partial URL—A Uniform Resource Locator (URL) that omits the protocol and server name and only specifies the path to the file relative to one another on the same server.

requirements—The list of customer needs for a Web site, such as search capability, tabbed menu navigation, specific color and branding requirements, or anything else that will create the desired outcome for the site.

Secure Sockets Layer (SSL)—Communications software that allows transmission of encrypted secure messages over the Internet.

shareware—Software that is distributed free so users can try it before they buy it. Users then can register the software for a relatively small fee compared to software produced commercially.

Shareware usually is developed by individuals or very small software companies, so registering the software is important.

site specification—The design document for your Web site.

Structured Query Language (SQL)—A programming language that lets you select information from a database.

taxonomy—A classification and naming of content in a hierarchical structure.

Uniform Resource Locator (URL)—The global address of documents and other resources on the Web.

Web analytics—The analysis of statistics that are gathered by Web servers.

Web server—A computer connected to the Internet that runs server software. The software lets the computer use the Hypertext Transfer Protocol (HTTP) to serve HTML files to Web browser clients.

wireframe—Web page mockups that represent page layouts for a Web site.

Review Questions

1. List three technology constraints that can affect the way a user views your Web site's content.

2. Consult your Web server administrator when you need to determine the _____ and _____ for your site.

3. Name two inconsistencies that can cause broken links when you upload your files to a Web server.

4. List three characteristics of filenames that vary by operating system.

5. The international standard for filenames often is called _____.

6. Which computer operating system is case sensitive?

7. Rename the following files so that they are compatible across all operating systems:

   My file.htm _____

   case:1.htm _____

   #3rdpage.htm _____

8. What is the default main page filename for a Web site?

9. What are the two types of URLs?

10. What are the four parts of a complete URL?

11. What type of URL links to another server?

12. What type of URL links within a server?

13. What affects the format of the URL for your Web site?

14. What is the benefit of purchasing a domain name?

15. Why should you never specify an absolute path in partial URLs?

16. What is the benefit of building a site with relative paths?

17. Files that reside in the same directory need only the _____ to refer to each other.

18. List two benefits of diagramming your site before you start coding.

19. How does a Web site become live?

20. List the four variables to consider when testing your Web site.

Hands-On Projects

1. Browse the Web and find a site you like. Write a brief statement of the Web site's goals.

2. Browse the Web and find Web sites that fit the following content types:

   a. Billboard

   b. Publishing

   c. Special interest

   d. Product support

   Write a short summary of how the content is presented in each Web site, and describe how each site focuses on its users' needs.

3. Browse the Web and find a site that does not contain a user survey form. Write a 10- to 15-question user survey that you would use on the site. Tailor the questions to the site's content and goals.

4. Browse the Web to find examples of the following site structures, and describe how the content fits the structure. Think about how the chosen structure adds to or detracts from the effectiveness and ease of navigation of the site. Determine whether the site provides sufficient navigation information. Print examples from the site, and indicate where the site structure and navigation information is available to the user.

   a. Linear

   b. Hierarchical

5. Browse the Web to find a site that uses more than one structure type, and describe why you think the site's content benefits from multiple structures. Consider the same questions as in Project 3-4.

6. Are there other structure types that are not described in this chapter? Find a site that illustrates a structure content not covered in this chapter. Create a flowchart for the site, and determine how it benefits from the different structure type.

7. Write a test plan for your Web site.
    a. Create a section for each design variable.
    b. Spell out the exact steps of the test and the different variables to be tested. State explicitly which browsers and versions should be used, and on which operating system. Detail the different screen resolutions and connection speeds. List the exact pages that should be tested.
    c. Walk through the test procedure to test its validity.
8. Write a sample user feedback questionnaire.
9. Write a maintenance plan for your Web site.
    a. Include a schedule of content updates for the different sections of the Web site.
    b. Include a schedule of design reviews.
    c. Plan for link maintenance.

Individual Case Project

Write a site specification for the site you defined in Chapters 1 and 2. Include as much information as possible from the project proposal you completed at the end of Chapter 1. Make sure to include a mission statement. Determine how you will measure the site's success in meeting its goals. Include a description of the intended audience. Describe how you will assess user satisfaction with the site. Include technological issues that may influence the site's development or function.

Prepare a detailed flowchart for your site using the preliminary flowchart you created at the end of Chapter 2. Create a filename for each page, using a consistent naming standard. Indicate all links between pages. Write a short summary that describes the flowchart. Describe why you chose the particular structure, how it suits your content, and how it benefits the user.

Use the page layouts you sketched in Chapter 2 and create wireframes. Download one of the free wireframe tools listed in this chapter, or use a drawing program to create the wireframes.

Team Case Project

Collaborate to write a site specification for the site you defined in Chapters 1 and 2. Include as much information as possible from the project proposal you completed at the end of Chapter 1. Make sure to include a mission statement. Determine how you will measure the site's success in meeting its goals. Include a description of the intended audience. Describe how you will assess user satisfaction with the site. Include technological issues that may influence the site's development or function.

Work individually to determine the information structure you think is optimal for the type of content your site will contain. Then meet and work as a team to determine the information structure using the best pieces of each team member's information structure plan. Prepare a detailed flowchart for your site using the preliminary flowchart you created at the end of Chapter 2. Create a filename for each page, using a consistent naming standard. Indicate all links between pages. Write a short summary that describes the flowchart. Describe why you chose the particular structure, how it suits your content, and how it benefits the user.

Use the page layouts you sketched in Chapter 2 and create wireframes. Download one of the free wireframe tools listed in this chapter, or use a drawing program to create the wireframes.

CHAPTER 4

Cascading Style Sheets

When you complete this chapter, you will be able to:

- Recognize the benefits of using CSS
- Build a basic style sheet
- Use inheritance to write simpler style rules
- Examine basic selection techniques
- Apply basic selection techniques
- Use class and id selectors
- Use the and elements
- Use other selectors

Cascading Style Sheets (CSS) let you control the display characteristics of your Web site. In this chapter, you examine the syntax of CSS and learn how to combine CSS rules with your HTML code. You start by examining the CSS style rules, and then apply them to build a basic style sheet. You learn selection techniques to apply a particular style declaration to an element in your document. You learn about inheritance and how it affects your styles. You learn about two elements, and , that were expressly created for use with CSS. More specific techniques include using the class attribute and id attribute to provide customized naming for your styles and applying the styles consistently across an entire Web site. You learn about using pseudo-selectors and pseudo-elements to select and apply CSS styles to links and to create special effects such as hovers, generated content, drop caps, and typographic effects.

Recognizing the Benefits of Using CSS

As you read in Chapter 1, Cascading Style Sheets offer many benefits to Web designers. CSS lets you separate style information from HTML, allowing you to provide style sheets for different destination media as your Web site requires. You can control the display characteristics of an entire Web site with a single style sheet, making maintenance and enhancements of display information a less taxing chore. You can express a wide range of style properties that increase the legibility, accessibility, and delivery of your content. You can build page layouts, either flexible or fixed, to suit your user and content needs. As you will see in this chapter and through the rest of the book, CSS is easy to learn and apply to your Web design projects.

The Evolution of CSS

Recall from Chapter 1 that Cascading Style Sheets were developed to standardize display information for Web pages and to enhance the separation of style and content. The first version of CSS, named CSS 1, was released in December 1996. CSS 1 contained properties to control fonts, color, text spacing and alignment, margins, padding, and borders. CSS 2, released in May 1998, added positioning, media types, and other enhancements. Even though CSS offered Web designers an easy-to-use and powerful method of controlling display properties, spotty browser support became a serious obstacle to the widespread adoption of the new style language. Finally, in May 2000, Internet Explorer (IE) 5 for the Macintosh became the first browser to fully support CSS 1.

Later, as CSS became more fully supported, inconsistencies across different browsers still made Web development a matter of testing and retesting to make sure that CSS style rules were implemented correctly. IE 6 had serious problems interpreting CSS correctly, and even though few people still use IE 6, its legacy of inconsistent support affects CSS usage to this day. It is only with the release of Internet Explorer 8 that all major browsers support CSS 2.1, which was released in 2005.

The latest release of CSS is CSS level 3 (CSS3). Although work started on CSS3 in 1998, this version is still not a complete recommendation from the W3C in 2010. (In this context, a **recommendation** is the final stage of development by the World Wide Web Consortium (W3C), and means the release has been reviewed and tested extensively.) In CSS3, the W3C has broken CSS into modules, each of which contains a specific set of CSS properties. Some features of CSS3 are supported by the major browsers, but consistent support is not yet a reality. You can choose to implement CSS3 features, but make sure to test and retest for compatibility. You may decide to implement CSS3 features realizing that some browsers, especially IE 8, may not offer users the same result that they will see in other browsers such as Firefox, Safari, and Opera. In some cases these differences may be negligible, and the CSS3 feature can be implemented without seriously affecting the user's view of your content. CSS3 properties are always indicated as such throughout this book so you can make informed decisions about which properties to implement.

Web sites such as *www.webdevout.net/browser-support-css* and *westciv.com* offer browser support cross-reference charts to help you determine which CSS properties are supported by different browser versions.

CSS Style Rules

In CSS, **style rules** express the style characteristics for an HTML element. A set of style rules is called a **style sheet**. Style rules are easy to write and interpret. For example, the following style rule sets all <p> elements in the document to blue text.

```
p {color: blue;}
```

A style rule is composed of two parts: a selector and a declaration. The style rule expresses the style information for an element. The **selector** determines the element to which the rule is applied. Selection is the key to working with CSS. As you will learn later in this chapter, CSS contains a variety of powerful selection techniques. By using the different types of selectors available, you build styles that affect an entire Web site or you can drill down to one specific paragraph or word in a Web page. The **declaration**, contained within curly brackets, details the exact property values. Figure 4-1 shows an example of a simple style rule that selects all <h1> headings and sets their color to red:

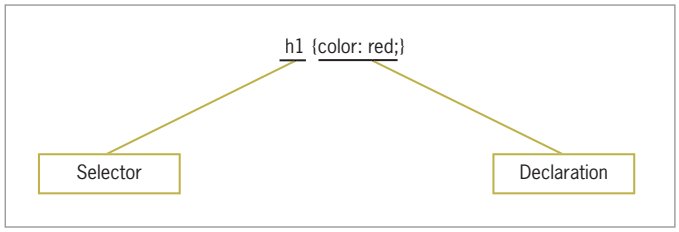


Figure 4-1 Style rule syntax

As illustrated in Figure 4-2, the declaration contains a property and a value. The **property** is a quality or characteristic, such as color, font size, or margin, followed by a colon (:). The **value** is the precise specification of the property, such as blue for color, 125% for font size, or 30 px (pixels) for margin, followed by a semicolon (;). CSS contains a wide variety of properties, each with a specific list of values. As you will see later in this chapter, you can combine selectors and property declarations in a variety of ways.

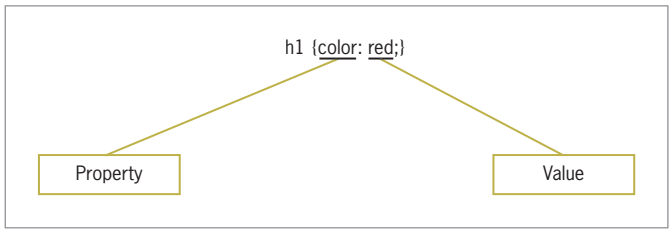


Figure 4-2 Property declaration syntax

This chapter uses a variety of CSS style rules as examples. Although you have not yet learned about their properties in detail, you will see that the CSS property names express common desktop publishing characteristics such as font-family, margin, text-indent, and so on. The property values sometimes use abbreviations such as *px* for pixel, percentages such as 200%, or keywords such as *bold*. You will learn about these properties and values in detail as you progress through this book.

Combining CSS Style Rules with HTML

You can combine CSS rules with HTML code in the following three ways:

- Inline style
- Internal style sheet
- External style sheet

Figure 4-3 shows the code for a single Web page with all three methods of combining CSS style rules with the HTML.

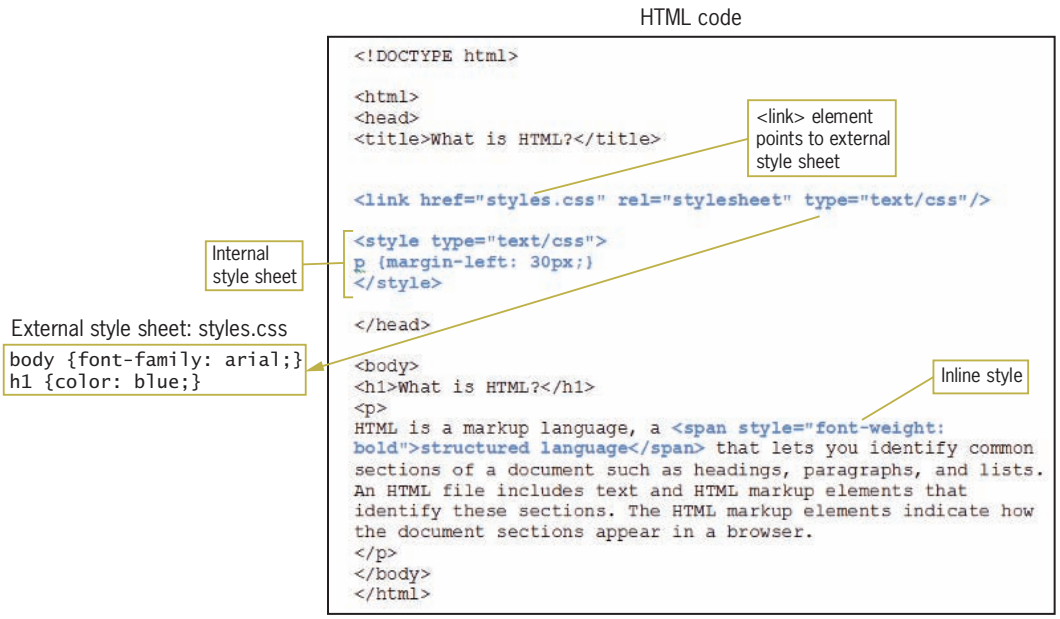


Figure 4-3 Three methods of combining CSS with HTML

As you review Figure 4-3, keep the following principles in mind:

- The <link> element in the document's <head> section points to an external style sheet named styles.css, which contains rules that set the body text to Arial and the <h1> heading to the color blue. This external style sheet can be linked from other documents as well, meaning these styles will be applied across multiple pages within the Web site.
- The internal style sheet affects only this one document in which it is contained, making all paragraph elements within the document have a 30-pixel left margin.
- The inline style within the paragraph affects only the span of words it surrounds, making the words *structured language* bold.

In this example, the styles rules from all three sources—external, internal, and inline—combine to create the finished look for this Web page. This flexibility in the application of style rules is an enormous benefit to Web designers, who can set styles that apply across an entire Web site using external style sheets, while maintaining control over individual pages and elements if necessary. Each method is discussed in detail in the following sections.

Using External Style Sheets

Placing style sheets in an external document lets you specify rules for multiple Web pages. This is an easy and powerful way to use style sheets because it lets you control the styles for an entire Web site with one style sheet file. Additionally, external style sheets are stored in the user's cache, so once downloaded, they are referenced locally for every file on your Web site, saving time for your user.

An external style sheet is simply a text document that contains the style rules. External style sheets have a .css extension. Here's an example of a simple external style sheet named styles.css:

```
h1 {color: white; background-color: green;}
h2 {color: red;}
```

The style sheet file does not contain any HTML code, just CSS style rules, because the style sheet is not an HTML document. It is not necessary to use the <style> element in an external style sheet.

Linking to an External Style Sheet

The **<link> element** lets you establish document relationships. It can only be used within the <head> section of a document. To link to an external style sheet, add the <link> element, as shown in the following code:

```
<head>
<title>Sample Document</title>
<link href="styles.css" rel="stylesheet" type="text/css" />
</head>
```

The <link> element in this code tells the browser to find the specified style sheet. The href attribute states the relative URL of the style sheet. The rel attribute specifies the relationship between the linked and current documents. The browser displays the Web page based on the CSS display information.

The advantage of the external style sheet is that you can state the style rules in one document and affect all the pages on a Web site. When you want to update a style, you only have to change the style rule once in the external style sheet.

Using Internal Style Sheets

Use the <style> element to create an internal style sheet in the <head> section of the document. Style rules contained in an internal style sheet only affect the document in which they reside. The following code shows a <style> element that contains a single style rule:

```
<head>
<title>Sample Document</title>
<style type="text/css">
h1 {color: red;}
</style>
</head>
```

In this code sample, note the type attribute to the <style> element. The value "text/css" defines the style language as Cascading Style Sheets.

Using Inline Styles

You can define the style for a single element using the style attribute, which is called an inline style.

```
<h1 style="color: blue">Some Text</h1>
```

You generally use the style attribute to override a style that was set at a higher level in the document, such as when you want a particular heading to be a different color from the rest of the headings on the page. (You'll learn more about overriding a style later in this chapter.) The style attribute is also useful for testing styles during development. You will probably use this method of styling an element the least, because it only affects one instance of an element in a document.

Writing Clean CSS Code

When you are creating external or internal style sheets, it is best to write CSS code that is consistent and easy to read. The flexibility of CSS syntax lets you write style rules in a variety of ways. For example, you've already seen style rules in this chapter that combine multiple declarations on the same line, as in the following code:

```
h1 {color: white; background-color: green;}
```

This technique is fine if you have simple styles, but style rules often contain many declarations that would be too hard to read in this format, especially when styles grow more complex, as shown in the following code:

```
p {font-family: arial, helvetica, sans-serif; font-size: 85%;
line-height: 110%; margin-left: 30px;}
```

This style rule is easier to read and maintain if you use a cleaner, more organized format as shown in the following code:

```
p {
   font-family: arial, helvetica, sans-serif;
   font-size: 85%;
   line-height: 110%;
   margin-left: 30px;
}
```

This format makes it easier to read the style rules, because all properties are consistently left-aligned with their values on the right.

Using Comments

CSS allows comments within the <style> element or in an external style sheet. CSS comments begin with the forward slash and asterisk characters (/*) and end with the asterisk and forward slash characters (*/). You can use comments in a variety of ways, as shown in the following code:

```
<style type="text/css">
/* This is the basic style sheet */
h1 {color: gray;} /* The headline color */
h2 {color: red;} /* The subhead color */
</style>
```

Comments provide documentation for your style rules. Because they are embedded directly in the style sheet, they provide immediate information to anyone who needs to understand how the style rules work. Comments are always useful, and you should consider using them in all of your code, whether as a simple reminder to yourself or as an aid to others with whom you work.

Activity: Building a Basic Style Sheet

In the following steps, you build and test a basic style sheet. Save your file and test your work in your browser as you complete each step. The new code to add is displayed in blue text. Refer to Figure 4-4 as you progress through the steps to see the results.

To build a basic style sheet:

1. Copy the **basic.htm** file from the Chapter04 folder provided with your Data Files to the Chapter04 folder in your work folder. (Create the Chapter04 folder, if necessary.)

2. In your browser, open **basic.htm**. When you open the Web page, it looks like Figure 4-4.

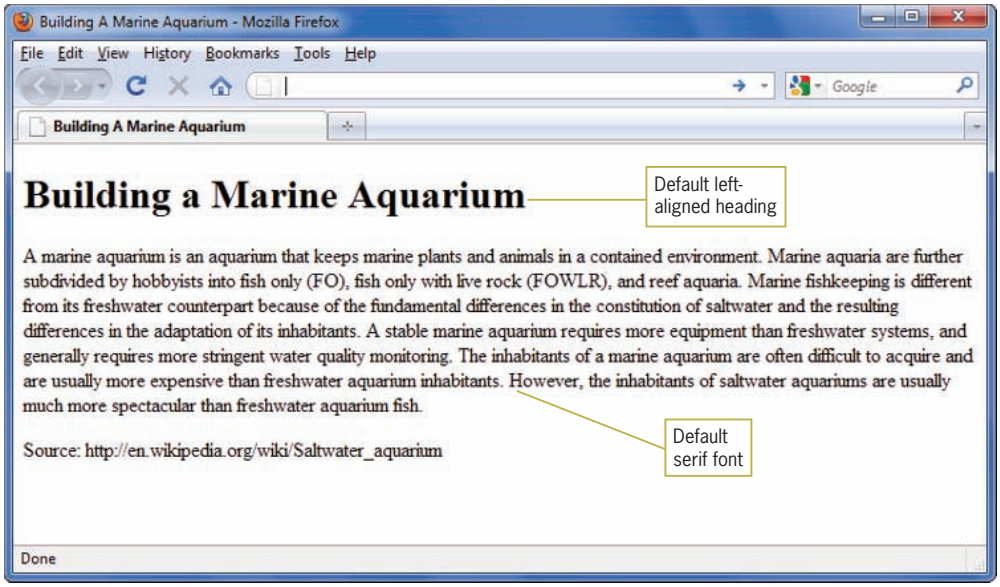


Figure 4-4 Original Web page with default styling

3. Open the **basic.htm** file in your HTML editor and examine the code. Notice that the file contains basic HTML code with no style information.

4. Add a <style> element in the <head> section to contain your style rules, as shown in blue in the following code. Leave a line or two of white space between the <style> tags to contain the style rules.

```
<head>
<style type="text/css">

</style>
</head>
```

5. Add a style rule for the <h1> element, as shown in blue in the following code. This style rule uses the text-align property to center the heading.

```
<head>
<style type="text/css">
h1 {text-align: center;}
</style>
</head>
```

6. Save the file as **basic.htm** in the Chapter04 folder in your work folder, and then reload the file in the browser. The <h1> element is now centered.

7. Add a style rule for the <p> element, shown in blue in the following code. This style rule uses the font-family property to specify a sans-serif font for the paragraph text. You will learn more about the font-family property in Chapter 5.

```
<head>
<style type="text/css">
h1 {text-align: center;}
p {font-family: sans-serif;}
</style>
</head>
```

8. Save the file and then reload it in the browser. Figure 4-5 shows the finished Web page. Notice that the <p> element is now displayed in a sans-serif typeface and the <h1> heading is centered.

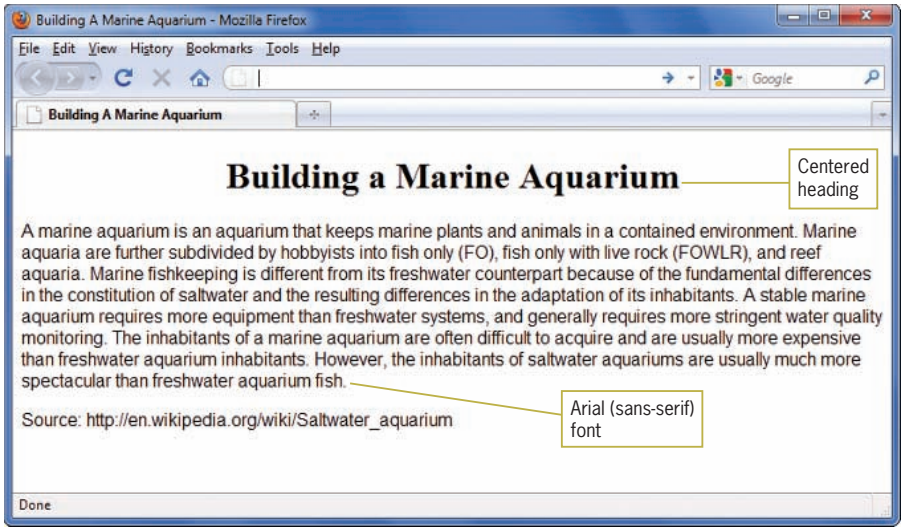


Figure 4-5 Finished Web page styled with CSS

Using Inheritance to Write Simpler Style Rules

The elements in an HTML document are structured in a hierarchy of parent and child elements. Figure 4-6 represents the structure of a simple HTML document.

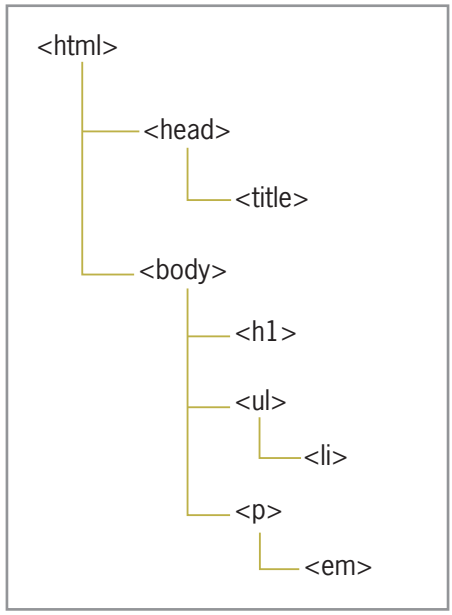


Figure 4-6 HTML document structure

Note the hierarchical structure of the elements. At the top, <html> is the parent element of the document. **Parent elements** contain nested elements called **child elements**. Both <head> and <body> are immediate child elements of <html>. Yet, <head> and <body> are parent elements as well, because they contain other nested elements. As you travel further down the document hierarchy, you find additional elements that are both parent and child elements, such as <p> and <ul>.

By default, most CSS properties inherit from parent elements to child elements, which is called **inheritance**. The CSS property descriptions in the following chapters and Appendix B list whether a property is inherited. Therefore, if you set a style rule for <ul> elements in the document shown in Figure 4-6, the <li> elements inherit the style rules for <ul>, unless you specifically set a rule for <li>.

You can style multiple document elements with just a few style rules if you let inheritance work for you. For example, consider the following set of style rules for a document.

```
<style type="text/css">
h1 {color: red;}
p  {color: red;}
ul {color: red;}
```

```
em {color: red;}
li {color: red;}
</style>
```

This style sheet sets the color to red for five different elements in the document. Inheritance lets you write a far simpler rule to accomplish the same results:

```
<style type="text/css">
body {color: red;}
</style>
```

This rule works because all of the elements are children of <body> and because all the rules are the same. It is much more efficient to write a single rule for the parent element and let the child elements inherit the style. Because <body> is the parent element of the content area of the HTML file, it is the selector to use whenever you want to apply a style across the entire document.

Examining Basic Selection Techniques

In this section you will review style rule syntax and learn about the following basic selection techniques:

- Using type selectors
- Grouping selectors
- Combining declarations
- Using descendant selectors

Using Type Selectors

As you learned previously, the selector determines the element to which a style declaration is applied. To review, examine the syntax of the style rule shown in Figure 4-7. This rule selects the <h1> element in the document and sets the text color to red.

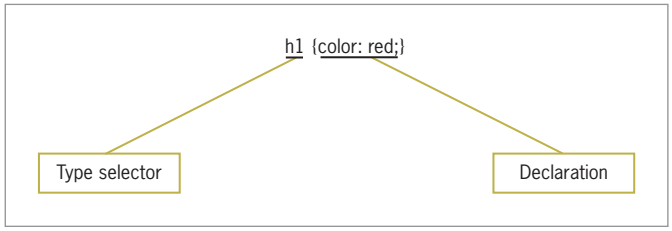


Figure 4-7 Style rule syntax

This rule uses a **type selector** to apply the rule to every instance of the element in the document. This is the simplest kind of selector, and many style sheets are composed primarily of type selector style rules, as shown in the following code:

```
body {color: gray;}
h2 {color: red;}
p {font-size: 85%;}
```

Grouping Selectors

To make your style rules more concise, you can group selectors to which the same rules apply. For example, the following style rules set the same declaration for two different elements—they set the color of <h1> and <h2> elements to red:

```
h1 {color: red;}
h2 {color: red;}
```

These two style rules can be expressed in a simpler way by separating the selectors with commas:

```
h1, h2 {color: red;}
```

Combining Declarations

In many instances you want to state multiple property declarations for the same selector. The following style rules set the <p> element to 12-point, blue text:

```
p {color: blue;}
p {font-size: 125%;}
```

These two style rules can be expressed in a simpler fashion by combining the declarations in one rule. The declarations are separated by semicolons:

```
p {
   color: blue;
   font-size: 125%;
}
```

Using Descendant Selectors

A descendant selector (sometimes known as a contextual selector) is based on the hierarchical structure of the elements in the document tree. This selector lets you select elements that are the descendants of other elements. For example, the following rule selects only elements that are contained within <p> elements. All other elements in the document are not affected.

```
p em {color: blue;}
```

Notice that the selector contains multiple elements, separated only by white space. You can use more than two elements if you prefer to choose more specific selection characteristics. For example, the following rule selects elements within <li> elements within <ul> elements only:

```
ul li em {color: blue;}
```

Using the Universal Selector

The **universal selector** lets you quickly select groups of elements and apply a style rule. The symbol for the universal selector is the asterisk (*). For example, to set a default color for all elements within a document, use the following rule:

```
* {color: purple;}
```

You can also use the universal selector to select all children of an element. For example, the following rule sets all elements within a element to a sans-serif typeface:

```
div * {font-family: sans-serif;}
```

Remember that the universal selector is always overridden by more specific selectors. The following style rules show a universal selector along with two other rules that have more specific selectors. In this example, the <h1> and <h2> rules override the universal selector for the <h1> and <h2> elements.

```
* {color: purple;}
h1 {color: red;}
h2 {color: black;}
```

Activity: Applying Basic Selection Techniques

In the following steps, you will build a style sheet that uses basic selection techniques. Save your file and test your work in your browser as you complete each step. The new code to add is displayed in blue text. Refer to Figure 4-8 as you progress through the steps to see the results.

To build the style sheet:

1. Copy the **oz.htm** file from the Chapter04 folder provided with your Data Files to the Chapter04 folder in your work folder.
2. Open the file **oz.htm** in your HTML editor, and save it as **oz1.htm** in the same location.
3. In your browser, open the file **oz1.htm**. When you open the file, it looks like Figure 4-8.

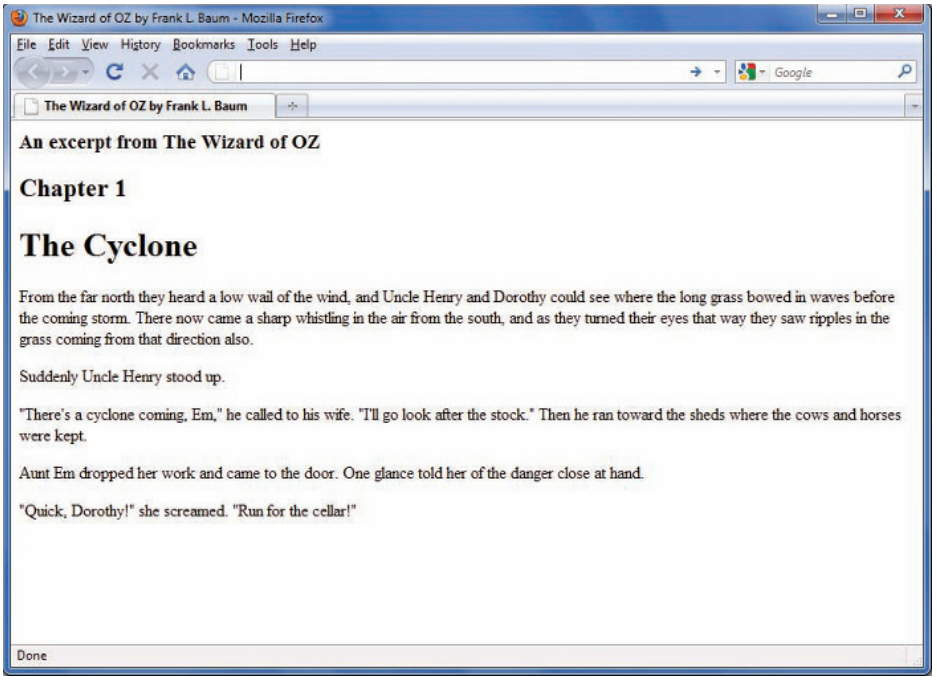


Figure 4-8 Original oz1.htm Web page

4. In your text editor, examine the page code. Notice that the file contains basic HTML code with no style information.

5. Add a <style> element in the <head> section to contain your style rules, as shown in blue in the following code. Leave a line or two of white space between the <style> tags to contain the style rules.

```
<head>
<title>The Wizard of OZ by Frank L. Baum</title>
<style type="text/css">

</style>
</head>
```

6. Write the style rule for the <h3> element. The requirements for this element are right-aligned text. The style rule looks like this:

```
<style type="text/css">
h3 {
   text-align: right;
}

</style>
```

7. Write the style rules for the <h1> and <h2> elements, which share some common property values. Both elements

have a left margin of 20 pixels (abbreviated as 20px) and a sans-serif font style. Because they share these properties, group the two elements to share the same style rule, as shown in the following code:

```
<style type="text/css">
h3 {
   text-align: right;
}

h1, h2 {
   margin-left: 20px;
   font-family: sans-serif;
}

</style>
```

8. Write an additional rule for the <h1> element. The <h1> element has two style properties that it does not share with <h2>, so a separate style rule is necessary to express the border and padding white space within the border. This rule uses the border shortcut property to specify multiple border characteristics—a 1-pixel border weight and solid border style. The padding-bottom property sets the border 5 pixels below the text.

```
<style type="text/css">
h3 {
   text-align: right;
}

h1, h2 {
   margin-left: 20px;
   font-family: sans-serif;
}

h1 {
   border-bottom: 1px solid;
   padding-bottom: 5px;
}

</style>
```

9. Write a style rule for the <p> elements so they have a 20-pixel left margin (to line up with the other elements on the page), Arial font style, and a font size of 120%.

```
<style type="text/css">
h3 {
   text-align: right;
}

h1, h2 {
   margin-left: 20px;
   font-family: sans-serif;
}
```

```
h1 {
   border-bottom: 1px solid;
   padding-bottom: 5px;
}

p {
   margin-left: 20px;
   font-family: arial;
   font-size: 120%;
}

</style>
```

Figure 4-9 shows the finished document with the style properties.

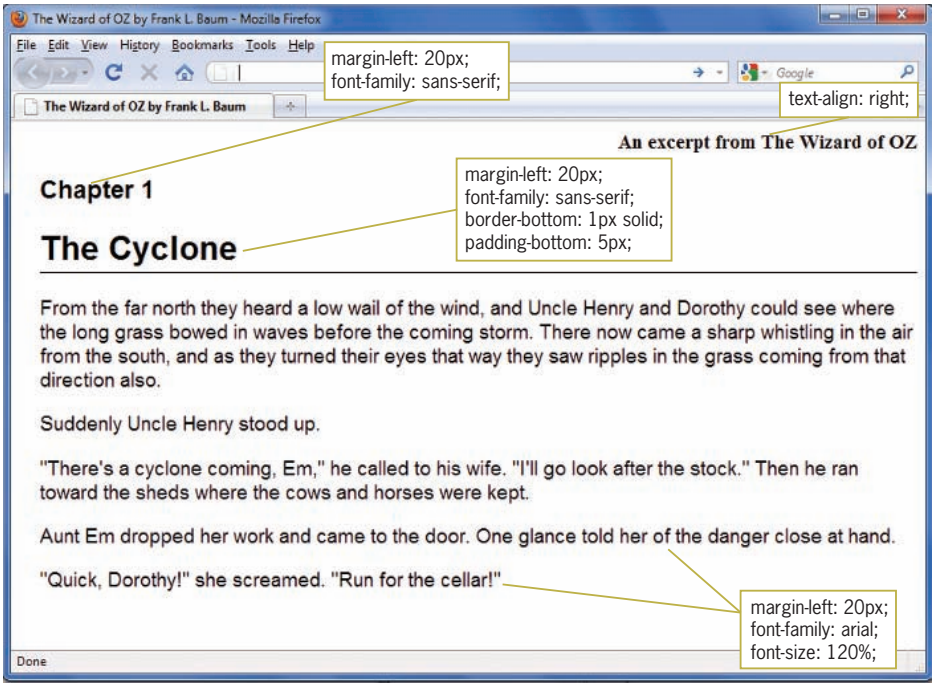


Figure 4-9 Finished Web page styled with CSS rules

Using Class and ID Selectors

This section describes CSS selection techniques that allow more than the basic element-based selection capabilities described in the previous section. You will learn to select elements of an HTML document using the following methods:

- The class selector
- The id selector

- The and elements
- The pseudo-class and pseudo-element selectors

Using the Class Selector

The class selector lets you write rules, give them a name, and then apply that name to any elements you choose. You apply the name using the class attribute, which is a common attribute that applies to any HTML element. Refer to Appendix A for descriptions of the common attributes. To apply a style rule to an element, you can add the class attribute to the element and set it to the name you have specified.

To create a class, declare a style rule. The period (**.**) flag character indicates that the selector is a class selector. Figure 4-10 shows an example of a rule with a class selector named *special*.

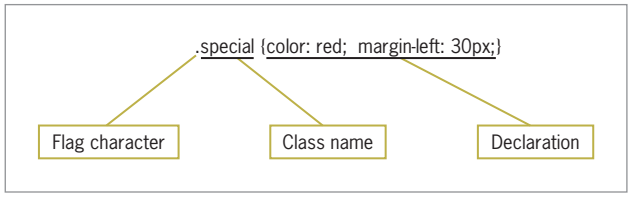


Figure 4-10 Class syntax

After writing the style rule, add it to the document by using the class attribute, as shown in the following code:

```
<h1 class="special">This heading will be red with a
30-pixel left margin.</h1>
```

The class attribute lets you select elements with greater precision. For example, read the following style rule:

```
p {font-size: 85%;}
```

This rule sets all <p> elements in the document to a font size of 85%. Now suppose that you want one <p> element in your document to have a different style characteristic, such as bold text. You need a way to specifically select that one paragraph. To do this, use a class selector. The following style rule sets the style for the class named *intro*:

```
.intro {font-size: 125%; font-family: sans-serif;}
```

The class selector can be any name you choose. In this instance, the class name *intro* denotes a special paragraph of the document. Now apply the rule to the <p> element in the document using the class attribute:

```
<p class="intro">This is the first paragraph of the
document. It has a different style based on the "intro"
class selector.</p>

<p>This is the second paragraph of text in the document.
It is a standard paragraph without a class attribute.</p>
```

Figure 4-11 shows the result of the style rule.

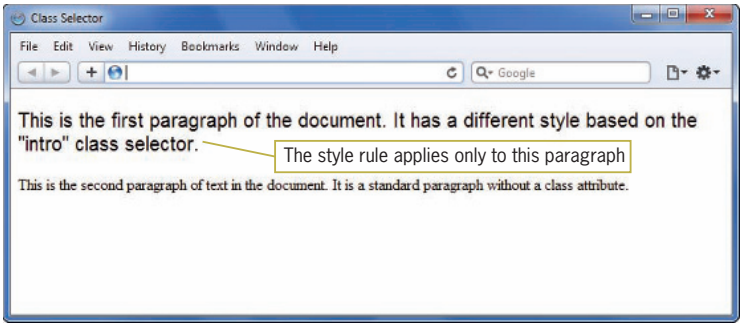


Figure 4-11 Styling with a class attribute

Making Class Selectors More Specific

Using the class attribute is a powerful selection technique, because it allows you to write style rules with names that are meaningful to your organization or information type. The more specific your class names become, the greater control you need over the way they are applied. In the preceding example, you saw a style rule named *intro* that was applied to a <p> element. However, the *intro* style can be applied to any element in the document, not just <p>. To solve this problem, you can restrict the use of the class attribute to a single element type.

For example, your organization might use a special style for a procedure heading, the heading that appears before steps in a training document. The style is based on an <h1> element, with a sans-serif font and left margin of 20 pixels. Everyone in your organization knows this style is named *procedure*. You can use this same style name in your style sheet, as shown in the following style rule:

```
.procedure {
   font-family: sans-serif;
   margin-left: 20px;
}
```

To use these rules in the document, you apply the class attribute, as shown in the following code:

```
<h1 class="procedure">Procedure Heading</h1>
```

This works well, but what happens if someone on your staff neglects to apply the classes properly? For the style rule to work, it must be applied to an <h1> element. To restrict the use of the class to <h1> elements, include a prefix for the class selector with the element to which you want it applied:

```
h1.procedure {
   font-family: sans-serif;
   margin-left: 20px;
}
```

These style rules restrict the use of the *procedure* style to <h1> elements only. If this style is applied to other elements it will not work.

Using the id Selector

The id attribute, like the class attribute, is an HTML common attribute. The difference between id and class is that it should refer to only one occurrence within a document. The id attribute has become the selector of choice when identifying layout sections of the page. The id attribute is perfect for this because generally only one layout section such as a footer or sidebar will occur per page. For example, you might want to specify that only one <p> element can have the id *copyright* and its associated style rule. Figure 4-12 shows a style rule that uses the id *copyright* as a selector.

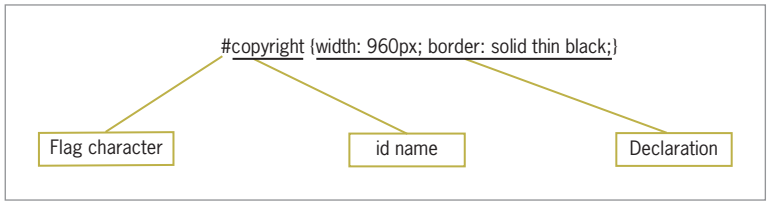


Figure 4-12 Using the id selector

Notice that the id selector uses a pound sign (#) flag character instead of the period you used with the class selector. You can apply the id value to the appropriate element in the document, in this example a <p> element:

```
<p id="copyright">This is the copyright information for
the page.</p>
```

The id value uniquely identifies this one <p> element as containing copyright information. For consistency in design, no other element in the document should share this exact id value.

Just like classes, you can make id selectors more specific by adding an element selector before the flag character, as shown in the following code:

```
p#copyright {
   font-family: times;
   text-align: center;
}
```

This style rule will only apply to a <p> element with the id set to *copyright*.

Using the and Elements

The (division) and (span of words) elements are designed to be used with CSS. They let you specify logical divisions within a document that have their own name and style properties. The difference between and is their element display type, which is described in more detail in Chapter 6. is a block-level element, and is its inline equivalent. Used with the class and id attributes, and let you effectively create your own element names for your HTML documents.

Working with Elements

You can use the element with the class and id attributes to create logical divisions on a Web page, such as headers, footers, columns, and so on. The element can contain other block level elements such as paragraphs or even other divisions. In modern Web design, divisions are the main content containers on a Web page, replacing the use of tables.

To create a customized division, declare it with a class or id selector in the style rule. The following example specifies a division with an id named *column* as the selector for the rule:

```
div#column {
   width: 200px;
   height: auto;
   padding: 15px;
   border: thin solid;
}
```

To apply this rule, specify the element in the document. Then use the id attribute to specify the exact type of division. In the following example, the code defines the element with the id named *column*:

```
<div id="column">This division displays as a column of
text in the browser window. This is one of the uses of the
division element as a page layout tool with CSS. You will
learn more about this in later chapters of this book. </div>
```

Figure 4-13 shows the result of the style rule.

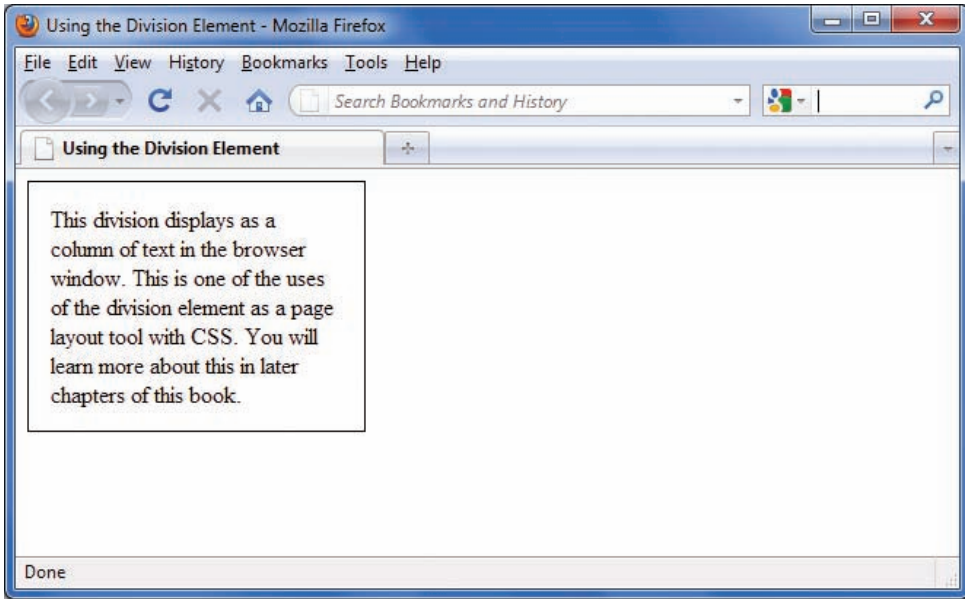


Figure 4-13 Using the element to create a column of text

Working with Elements

The element lets you specify inline elements within a document that have their own name and style properties. Inline elements reside within a line of text, like the or element. You can use with a class or id attribute to create customized inline elements.

To create a span, declare it within the <style> element first. The following example specifies a span named *logo* as the selector for the rule:

```
span.logo {
   color: white;
   background-color: black;
}
```

Next, specify the element in the document. Then use the class attribute to specify the exact type of span. In the following example, the code defines the element as the class named *logo*.

```
<p>Welcome to the <span class="logo">Wonder Software</span>
   Web site.</p>
```

Figure 4-14 shows the result of the style rule.

Figure 4-14 Using the element

Using Other Selectors

Besides class and id selectors, you can use attribute selectors, pseudo-class and pseudo-element selectors, and CSS3 selectors.

Using Attribute Selectors

Attribute selectors let you select an element based on whether the element contains an attribute. You can also choose an element based on a specific value the attribute contains.

Attribute selectors match against attributes and their values. In the following code, the element has three attributes: src, title, and alt.

```
<img src="images/home.gif" title="home" alt="Home
navigation button" />
```

Using attribute selectors, you could choose this based on the presence of the title attribute, as in the following code:

```
img[title]  {border-color: red;}
```

Or, you could choose this based on the value that the title attribute contains, as shown:

```
img[title=home]  {border-color: red;}
```

Attribute selectors come in handy when there are multiple elements that share the same characteristics, sometimes only differing by the values of the attributes they contain. Table 4-1 lists the CSS 2.1 attribute selectors, which would be more commonly supported by most browsers. CSS3 selectors are described later in this chapter.

Syntax	Description	Example
[attribute]	Select when the element contains the named attribute; the attribute value is not matched	p[title] {color: blue;} matches: <p title="opening"> <p title="closing">
[attribute=value]	Select when the element contains the named attribute and specific value	p[title=footer] matches: <p title="footer">
[att~=val]	Select when the attribute contains the value in a list of white-space separated values	p[att~=copyright] matches: <p type="copyright trademark">
[att|=val]	Select when an attribute contains the exact value or begins with the value	p[att|en] matches: <p lang="english">

Table 4-1 CSS 2.1 Attribute Selectors

Using Pseudo-Class and Pseudo-Element Selectors

Pseudo-class and pseudo-element selectors let you express style declarations for characteristics of a document that are not signified with the standard HTML elements. **Pseudo-classes** select elements based on characteristics other than their element name. For example, assume that you want to change the color of a new or visited hypertext link. No HTML element directly lets you express these characteristics of the <a> element. With CSS, you can use the pseudo-class selector to change the link color.

Pseudo-elements let you change other aspects of a document that are not classified by elements, such as applying style rules to the first letter or first line of a paragraph. For example, you might want to create a drop initial or drop capital that extends below the line of type, or make the first line of a paragraph all uppercase text. These are common publishing design techniques that are not possible with standard HTML code. With CSS you can use the :first-letter and :first-line pseudo-elements to add these two style characteristics to your documents.

Using the Link Pseudo-Classes

The link pseudo-classes let you change the style characteristics for four different hypertext link states, as described in Table 4-2.

Pseudo-Class	Description
:link	Selects any unvisited link that the user has not clicked or is not hovering over with their mouse pointer
:visited	Selects any link that your user has already visited
:hover	Selects any link that your user is pointing to with the mouse pointer
:active	Selects a link for the brief moment that your user is actually clicking the link

Table 4-2 Link Pseudo-Classes

The following rules change the colors of the hypertext links in a Web page:

```
a:link {color: red;}
a:visited {color: green;}
```

Note that the colon (:) is the flag character for a pseudo-class.

Because of the specificity of the pseudo-class selectors, you should always place your link pseudo-class in the following order:

1. Link
2. Visited
3. Hover
4. Active

This order is sometimes abbreviated to LVHA. You do not have to use all of the different link states, so if you skip one make sure the others follow the correct order. You will learn about specificity later in this chapter.

Whether you choose to change your hypertext link colors depends on the design of your site and the needs of your users. Remember that many Web users are comfortable with the default underlining, and that color alone may not be enough to differentiate links from the rest of your text.

Using the :hover Pseudo-Class

The :hover pseudo-class lets you apply a style that appears when the user points to an element with a pointing device. This is a useful navigation aid to add to the <a> element, with the result that the link appears highlighted when the user points to it with the mouse. The following style rule shows the :hover pseudo-class with the <a> element as the selector.

```
a:hover {background-color: yellow;}
```

This style rule changes the background color of the link to yellow, which is an effective highlight color. Figure 4-15 shows the results of this style rule.

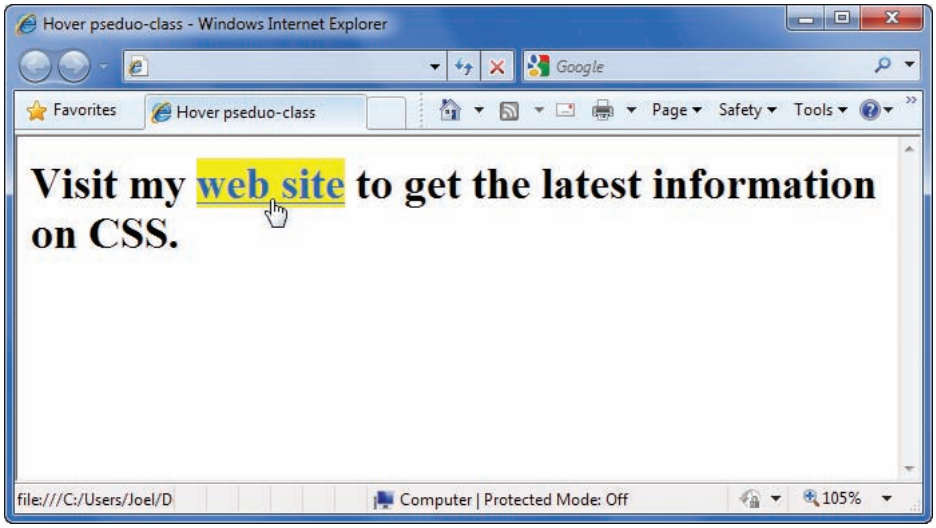


Figure 4-15 The :hover pseudo-class is activated by the mouse pointer

Using the :first-letter Pseudo-Element

Use the :first-letter pseudo-element to apply style rules to the first letter of any element. This lets you create interesting text effects, such as initial capitals and drop capitals, which are usually set in a bolder and larger font. Initial capitals share the same baseline as the rest of the text, while drop capitals extend down two or more lines below the text baseline. To apply :first-letter to build an initial capital, specify a style rule like the following:

```
p:first-letter {
  font-weight: bold;
  font-size: 200%;
}
```

This creates a first letter that is bold and twice the size of the <p> font. For example, if the <p> element has a font size of 12 points, the initial cap will be 24 points.

To make sure that this rule applies only to one paragraph, rather than every paragraph in the document, a class name needs to be added to the style rule. To solve this problem, add a class name, such as *initial*, to the rule, as shown in Figure 4-16.

The: hover pseudo- class works on other elements as well; for example, you can use hover to highlight any text that a user points to with their mouse pointer.

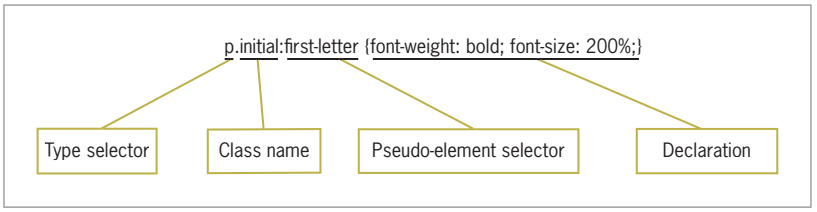


Figure 4-16 Using a class selector with a :first-letter pseudo-element

This style rule affects only <p> elements with the class value of *initial*, as shown in the following code:

```
<p class="initial">From the far north they heard a low
wail of the wind, and Uncle Henry and Dorothy could see
where the long grass bowed in waves before the coming
storm. There now came a sharp whistling in the air from
the south, and as they turned their eyes that way they saw
ripples in the grass coming from that direction also.</p>
```

Figure 4-17 shows the result of the style rule.

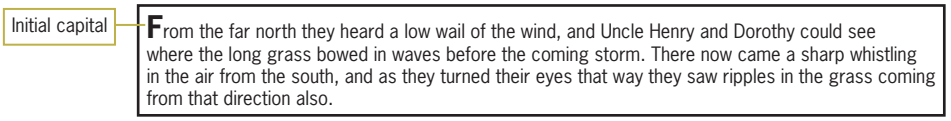


Figure 4-17 Initial capital styled with the :first-letter pseudo-element

You can make the initial capital a drop capital by adding the float property to the rule, which allows the letter to extend downward. The float property is described in Chapter 6. Here is a :first-letter style rule with the float property added:

```
p.dropcap:first-letter {
   font-weight: bold;
   font-size: 200%;
   float: left;
}
```

Notice that the class has been changed to signify that this first letter is a drop capital. Remember, you can set the class attribute to any naming value that makes sense to you.

This style rule affects only <p> elements with the class value of *dropcap*, as shown in the following code:

```
<p class="dropcap">From the far north they heard a low
wail of the wind, and Uncle Henry and Dorothy could see
where the long grass bowed in waves before the coming
```

```
storm. There now came a sharp whistling in the air from
the south, and as they turned their eyes that way they saw
ripples in the grass coming from that direction also.</p>
```

Figure 4-18 shows the result of the new style rule.

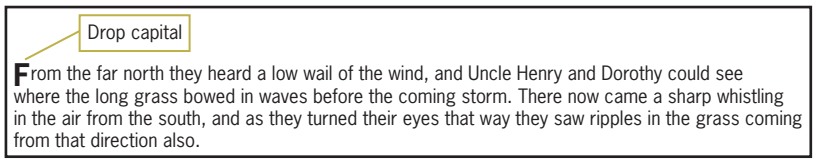


Figure 4-18 Drop capital using the :first-letter pseudo-element

The :first-letter pseudo-element can only be applied to a block-level element. In addition, only the following properties can be applied to the :first-letter selector:

- Font properties
- Color properties
- Background properties
- Margin properties
- Padding properties
- Word-spacing
- Letter-spacing
- Text-decoration
- Vertical-align
- Text-transform
- Line-height
- Text-shadow
- Clear

Using the :first-line Pseudo-Element

The :first-line pseudo-element works in much the same way as :first-letter, except for the obvious difference that it affects the first line of text in an element. For example, the following rule sets the first line of every <p> element to uppercase letters:

```
p:first-line {text-transform: uppercase;}
```

The problem with this code is that it affects every <p> element in the document. As you saw in the preceding :first-letter selector,

you can add a class attribute to more narrowly define the application of the :first-line style:

```
p.introduction:first-line {text-transform: uppercase;}
```

This rule transforms to uppercase the first line of the <p> element that contains the following code:

```
<p class="introduction">From the far north they heard a
low wail of the wind, and Uncle Henry and Dorothy could
see where the long grass bowed in waves before the coming
storm. There now came a sharp whistling in the air from
the south, and as they turned their eyes that way they saw
ripples in the grass coming from that direction also.</p>
```

Figure 4-19 shows the results of the style rule.

FROM THE FAR NORTH THEY HEARD A LOW WAIL OF THE WIND, AND UNCLE Henry and Dorothy could see where the long grass bowed in waves before the coming storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.

Figure 4-19 First line transformation using the :first-line pseudo-element

The :first-line pseudo-element can only be applied to a block-level element. In addition, only the following properties can be applied to :first-line. Notice that :first-line does not support padding, margin, or border properties:

- Font properties
- Color properties
- Background properties
- Word-spacing
- Letter-spacing
- Text-decoration
- Text-transform
- Line-height
- Text-shadow
- Clear

Using the :before and :after Pseudo-Elements

The :before and :after pseudo-elements let you insert content in your Web page that is created by the style sheet rather than contained in your document text. Your Web page content may include

terms that must be used repeatedly. These can be inserted by a style sheet rather than having to enter them over and over in your content. A good example of this is having the word *Figure* in the title of all of your figures. For example, this style rule will insert the word *Figure* followed by a colon before any <p> text that has the class figtitle:

```
p.figtitle:before {content: "Figure: "}
```

The :before pseudo-element places generated content before the selected element, and the :after pseudo-element places generated content after the selected element. Here's an example that uses :before to add generated content to a glossary. Style rules select the elements and apply generated content based on the class name.

```
p.term:before {content: "Term: "; font-weight: bold;}
p.definition:before {content: "Definition: "; font-weight:
bold;}
```

These style rules apply to the following HTML code. Figure 4-20 shows the results of the following code.

```
<p class="term">Quasar</p>
<p class="definition"> A quasi-stellar radio source
(quasar) is a very energetic and distant galaxy with
an active galactic nucleus. They are the most luminous
objects in the universe.</p>
```

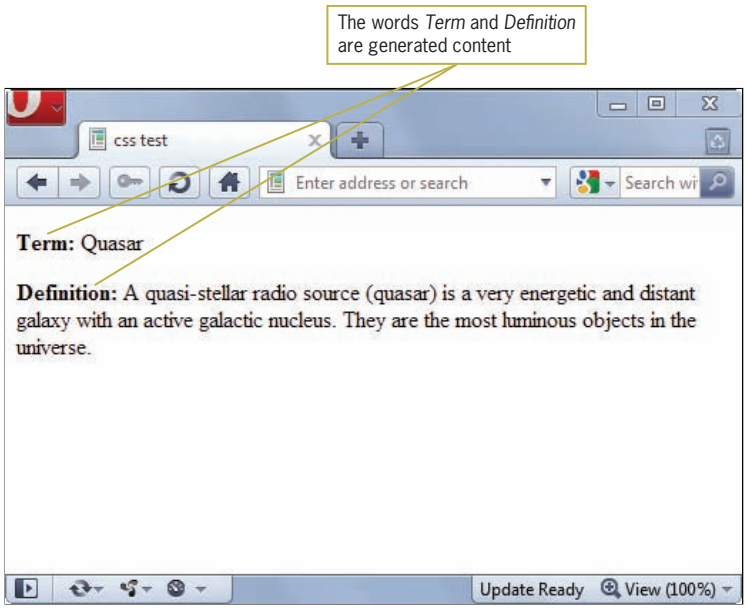


Figure 4-20 Results of using the :before pseudo-element

Understanding How the Cascade Affects Style Rules

One of the fundamental features of CSS is that style sheets **cascade**. This means that multiple style sheets and style rules can apply to the same document. HTML authors can attach a preferred style sheet, while the reader might have a personal style sheet to adjust for preferences such as human or technological handicaps. However, only one rule can apply to an element. The CSS cascading mechanism determines which rules are applied to document elements by assigning a weight to each rule based on the following three variables:

- Specificity of the selector
- Order of the rule in the style sheet
- Use of the !important keyword

Determining Rule Weight by Specificity

Another method of determining style rule weight is the specificity of the rule's element selector. Specificity determines which rule, if there is a conflict, applies to your HTML elements. Rules with more specific selectors take precedence over rules with less specific selectors.

Specificity is actually a complex calculation that is performed by the browser to determine which rule takes precedence, based on adding the weight values of the possible style rules that affect an element.

Specificity conflicts normally occur when multiple style sheets affect a Web page, which is common in larger content management systems, and when compounded selectors are used to narrowly target a specific element on a page. For example, the following style rules target a <p> element. The first applies to all <p> elements on the page, and the second applies to <p> elements that are specifically within a element. The more specific descendant selector determines the outcome, so the <p> elements contained within a will be blue.

```
p {color: red:}
div p {color: blue;}
```

This is a very simple example of specificity to illustrate the concept. The following general rules make it easier to understand which style rule takes precedence.

- Inline styles, using the style attribute, have the greatest weight.
- Styles with id selectors override styles with class selectors.
- Class selectors override simple type selectors.

Determining Rule Weight by Order

CSS applies weight to a rule based on its order within a style sheet. Rules that are included later in the style sheet order take precedence over earlier rules. Examine the following style rules for an example:

```
body {color: black;}
h1 {color: red;}
h1 {color: green;}
```

In this example, <h1> elements in the document appear green because the last style rule specifies green as the color. This also applies to internal and external style sheets. Internal style sheet rules have greater weight than external style sheet rules

Determining Rule Weight with the !important Keyword

A conflict can arise when multiple rules apply to the same element. By default, rules that are closest to the element, or more specific, always take precedence The **!important** keyword ensures that a particular rule will always apply. The following style sheet states a rule for <p> elements that sets the font family to arial, regardless of other rules that might apply to the element:

```
<style type="text/css">
p {font-family: arial !important;}
</style>
```

The !important keyword also allows you to specify that a rule should take precedence no matter what the order in the style sheet. Examine the following style rules for an example:

```
body {color: black;}
h1 {color: red !important;}
h1 {color: green;}
```

In this example, <h1> elements in the document will appear red, even though the rule that sets the color to green would normally take precedence because of its order in the document.

However, it is always best to use the !important keyword sparingly since it defeats the flexible nature of CSS.

CSS3 Selectors

CSS3 adds more new selectors that let you narrow down the exact element you want to select with even greater precision. Many of the new selectors let you choose elements based on where they reside in the document structure, letting you make selections such as the first or last paragraph on every page, or setting different

styles for odd or even rows of a table or list. Most browsers support these selectors, but make sure to test carefully for compatibility. In this section, you will learn about the more commonly used CSS3 selectors. You will find a complete list in Appendix A.

You can view CSS3 selector compatibility charts at:

www.css3.info/modules/selector-compat.

CSS3 includes a number of new types of selectors, including the following:

- Substring matching attribute
- Structural pseudo-class
- UI element states

Substring Matching Attribute Selectors

These selectors match parts of an attribute value, such as the beginning or ending of a string of text. Table 4-3 lists the syntax and description for substring matching attribute selectors.

Selector Syntax	Description
p[att^="val"]	Matches any p element whose att attribute value begins with "val"
p[att$="val"]	Matches any p element whose att attribute value ends with "val"
p[att*="val"]	Matches any p element whose att attribute value contains the substring "val"

Table 4-3 Substring Matching Attribute Selectors

For example, the following rule selects div elements whose class attribute begins with the word *content*:

```
div[class^="content"]
```

Structural Pseudo-Class Selectors

Earlier in this chapter you read how an HTML document is a tree structure of parent and child elements. The structural pseudo-classes let you select elements based on where they reside in the document tree. Table 4-4 lists the syntax and description for structural pseudo-class selectors.

Selector Syntax	Description
:root	Matches the document's root element; in HTML, the root element is always the HTML element
p:nth-child(n)	Matches any <p> element that is the *n*th child of its parent
p:nth-last-child(n)	Matches any <p> element that is the *n*th child of its parent, counting from the last child
p:nth-of-type(n)	Matches any <p> element that is the *n*th sibling of its type
p:nth-last-of-type(n)	Matches any <p> element that is the *n*th sibling of its type, counting from the last sibling
p:last-child	Matches any <p> element that is the last child of its parent
p:first-of-type	Matches any <p> element that is the first sibling of its type
p:last-of-type	Matches any <p> element that is the last sibling of its type
p:only-child	Matches any <p> element that is the only child of its parent
p:only-of-type	Matches any <p> element that is the only sibling of its type
p:empty	Matches any <p> element that has no children (including text nodes)

Table 4-4 Structural Pseudo-Class Selectors

For example, if you wanted to style the last <p> element on a Web page, you can use this style rule:

```
p:last-child {border-bottom: solid thin black;}
```

UI Element States Selectors

The UI element states selectors let you choose an element based on its state of user interaction. Table 4-5 lists the syntax and description of UI element states selectors.

Selector Syntax	Description
:enable	Matches any interface element, such as a form control, that is in an enabled state
:disabled	Matches any interface element, such as a form control, that is in a disabled state
:checked	Matches any option button or check box that is in a selected state

Table 4-5 UI Element States Selectors

For example, you could highlight a selected option button with the following style rule:

```
input:checked {background-color: yellow;}
```

Chapter Summary

This chapter presents the basic syntax of the CSS language. You learned about the different methods of selecting elements and applying style rules in a variety of ways. You saw that the CSS basic selection techniques are often powerful enough to handle most document styling. You learned that the class and id attributes let you create naming conventions for styles that are meaningful to your organization or information type. As you will see in the upcoming chapters, CSS is an easy-to-use style language that lets you gain visual control over the display of your Web content.

- CSS rules can be combined with your HTML code in a number of ways. CSS rules are easy to write and read.
- CSS uses inheritance and cascading to determine which style rules take precedence.
- Basic style rules let you apply style rules based on standard element selectors. You can combine the selectors and declarations to create more powerful style expressions. You can also select elements based on the contextual relationship of elements in the document tree.
- You can use the class and id attribute selectors, which are often paired with the and HTML elements. You can create rules, name them, and apply them to any element you choose.
- The pseudo-class and pseudo-element selectors let you change the color and styling of hypertext links and the effect elements of a document, such as first line and first letter, that are not signified with the standard HTML elements.

Key Terms

!important—A CSS keyword that lets the user override the author's style setting for a particular element.

cascade—Style sheets originate from three sources: the author, the user, and the browser. The cascading feature of CSS lets these multiple style sheets and style rules interact in the same document.

child element—An HTML element contained within another element.

declaration—The declaration portion of a style rule consists of a property name and value. The browser applies the declaration to the selected element.

inheritance—The order of CSS rules dictating that child elements inherit rules from parent elements.

<link> element—An HTML element that lets you establish document relationships, such as linking to an external style sheet.

parent element—An HTML element that contains child elements.

property—A quality or characteristic stated in a style rule, such as color, font-size, or margin. The property is a part of the style rule declaration.

pseudo-class—An element that selects elements based on characteristics other than their element name.

pseudo-element—An element that lets you change other aspects of a document that are not classified by elements, such as applying style rules to the first letter or first line of a paragraph.

recommendation—The final stage of development by the W3C, indicating that a technology release has been reviewed and tested extensively.

selector—The part of a style rule that determines which HTML element to match. Style rules are applied to any element in the document that matches the selector.

style rule—The basic unit of expression in CSS. A style rule is composed of two parts: a selector and a declaration. The style rule expresses the style information for an element.

style sheet—A set of style rules that describes a document's display characteristics. There are two types of style sheets: internal and external.

type selector—A CSS selector that applies a rule to every instance of the element in this document.

universal selector—A selector that lets you quickly select groups of elements and apply a style rule.

value—The precise specification of a property in a style rule, based on the allowable values for the property.

Review Questions and Exercises

1. What are the two parts of a style rule?
2. What are the three ways to combine CSS rules with your HTML code?
3. List two reasons to state a style using the style attribute.
4. What are the advantages of using an external style sheet?
5. What is the inheritance default for CSS rules?
6. What is the benefit of the !important declaration?
7. Write a basic style rule that selects <h1> elements and sets the color property to red.
8. Add the <p> element as an additional selector to the rule you created for Question 7.
9. Add a font-size property to the rule and set the size to 14 points.
10. Write a style rule that selects <ul> elements only when they appear within <p> elements and set the color property to red.
11. Write the style rule for a class selector named note. Set the font-weight property to bold.
12. Restrict the rule you developed for Question 11 so it can be used only with <p> elements.
13. What is the difference between and ?
14. Write a style rule that sets the default document text color to red.
15. What is the advantage of working with the class attribute?
16. What element does this selector choose?

```
p ul li
```

17. What element does this selector choose?

```
div p *
```

18. What element does this selector choose?

```
p.warning
```

19. What is the advantage of working with the id attribute?

20. Write a style rule that applies a yellow background color to <a> elements when the user points the mouse to a hypertext link.

Hands-On Projects

1. Visit the World Wide Web Consortium Web site and find the CSS Techniques for Web Content Accessibility Guidelines (*www.w3.org/TR/WCAG10-CSS-TECHS*). Write an essay describing the benefits of using CSS for accessibility and the primary CSS features you would use to ensure accessible content on the Web.

2. By yourself or with a partner, choose a mainstream publishing Web site, such as a newspaper or periodical site. Examine the style characteristics of the site. What common styles can be applied across the site, such as headings, paragraphs, and bylines? Write an analysis of the site's style requirements, and list the styles you would include in the site's style sheet.

3. In this project, you will have a chance to test a few simple style rules on a standard HTML document and view the results in your browser.

   a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains <body>, <h1>, and <p> elements. Save the file as **csstest1.htm** in the Chapter04 folder in your work folder.

   b. Add a <style> element to the <head> section, as shown in the following code.

```
<head>
<title>CSS Test Document</title>
<style type="text/css">

</style>
</head>
```

c. Add a style rule that uses *body* as a selector and sets the color property to green, as shown in the following code:

```
<style type="text/css">
body {color: green;}
</style>
```

d. Save **csstest1.htm** and view it in the browser. All of the document text should now be green.

e. Now add a style rule that sets <h1> elements to be displayed in black:

```
<style type="text/css">
body {color: green;}
h1 {color: black;}
</style>
```

f. Save **csstest1.htm** and view the results in the browser.

g. Finally, add a style rule that sets a margin for <p> elements to 30 pixels:

```
<style type="text/css">
body {color: green;}
h1 {color: black;}
p {margin: 30px;}
</style>
```

h. Save **csstest1.htm** and view the results in the browser.

4. In this project, you will have a chance to test a few basic selection techniques on a standard HTML document and view the results in your browser. Save the file and view it in your browser after completing each step.

a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains <body>, <h1>, <p> elements, and so on. Save the file in the Chapter04 folder in your work folder as **csstest2.htm**.

b. Add a <style> element to the <head> section, as shown in the following code:

```
<head>
<title>CSS Test Document</title>
<style type="text/css">

</style>
</head>
```

c. Write a style rule that uses *body* as a selector and sets the color property to the color of your choice.

d. Find two elements on the page, such as <h1> and <h2>, which can share the same characteristics. Write a single style rule that applies to both elements. Set the color property to red and the margin property to 20px.

e. Find one element that contains another, such as a <b> or <q> element within a <p> element. Write a descendant selector rule that affects the contained element and sets the color property to green.

5. In this project, you will have a chance to test a few advanced selection techniques on a standard HTML document and view the results in your browser. Save the file and view it in your browser after completing each step.

   a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains the elements: <body>, <h1>, <p>, and so on. Save the file in the Chapter04 folder in your work folder as **csstest3.htm**.

   b. Add a <style> element to the <head> section, as shown in Exercise 4.

   c. Write a rule for a class selector named *heading*. Set the color property to red and the font-size property to 200%. Apply the heading class to an <h1> or <h2> element in the document.

   d. Write a rule for a class selector named *emphasis*. Set the color property to blue and the background-color property to yellow. In the document, add a element to a span of words that you want to highlight. Apply the emphasis class to the element.

Individual Case Project

Revisit your project proposal and the site specifications you created in Chapter 3. How will you implement Cascading Style Sheets into your project Web site? In the next few chapters, you will learn how to control typography, white space, borders, colors, and backgrounds with CSS. Think about each of these style characteristics

and how you will apply them to your page designs. In addition, make a list of possible class names you might use to identify your content. For example, consider using class names for the following page characteristics, as well as creating some of your own:

- Body copy
- Header (possibly different levels)
- Footer

Team Case Project

Revisit your project proposal and the site specifications you created in Chapter 3. Each team member is responsible for individual templates, such as the home page template, the section page template, and the article level page template.

In the next few chapters, you will learn how to control typography, white space, borders, colors, and backgrounds with CSS. Decide how you will handle these style characteristics in your Web site. Each team member should create a suggested list of styles and naming conventions for use in your site.

For example, you might have top-level, secondary-level, and tertiary-level headings. What names will you use for consistency throughout the site? You might want to name these *A-head*, *B-head*, and so on. Also, you will need to segregate and name the different copy styles in your site. For example, you might have a different body copy style for the main page than you have on a secondary section or on reading-level pages. Think about the different style properties that you will be able to manipulate (you can look these up in Appendix B) and how you will consistently use, manage, and name these style characteristics for your project Web site. Meet as a team to review each member's ideas and come to an agreement on the proposed styles and naming conventions.

CHAPTER 5

Web Typography

When you complete this chapter, you will be able to:

- Understand type design principles
- Understand Cascading Style Sheets (CSS) measurement units
- Use the CSS font properties
- Use the CSS text-spacing properties
- Build a font and text properties style sheet
- Customize bulleted and numbered lists

The type choices you make for your site provide the foundation for the clear communication of your content. The consistent use of type to express the hierarchy of your content provides valuable information cues to the reader, and the choices you make to enhance text legibility affect the usability of your Web site. Recent innovations provide powerful tools for increased typographic choices and control. In the past, Web typography meant having to use too many <font> tags and lots of text as graphics. Today, Cascading Style Sheets offers a potent style language, allowing you to manipulate a variety of text properties to achieve professional, effective results, all without resorting to graphics that add download time. The addition of new type properties in CSS3 will allow Web designers even more typographic choices as commercial fonts become available for use.

Understanding Type Design Principles

Type can flexibly express emotion, tone, and structure. Most of the type principles that apply to paper-based design apply to the Web as well. For example, it is possible to go overboard by using too many typefaces and sizes. Just because you have many typefaces at your disposal does not mean you should use them all. In addition, designing for the Web actually restricts your font choices to those your users have installed on their computers.

In strict typography terms, a **typeface** is the name of the type, such as Times New Roman or Futura Condensed. A **font** is the typeface in a particular size, such as Times Roman 24 point. For the most part, on the Web the two terms are interchangeable.

As you work with type, consider the following principles for creating an effective design:

- Choose fewer fonts and sizes.
- Use available fonts.
- Design for legibility.
- Avoid using text as graphics.

Choose Fewer Fonts and Sizes

Your pages will look cleaner when you choose fewer fonts and sizes of type. Decide on a font for each level of topic importance, such as page headings, section headings, and body text. Communicate the hierarchy of information with changes in the size, weight, or color of the typeface. For example, a page heading should have a larger, bolder type, while a section heading would appear in the same typeface, only lighter or smaller.

Pick a few sizes and weights in a type family. For example, you might choose three sizes: a large one for headings, a smaller size

for subheadings, and your body text size. You can vary these styles by changing the weight; for example, bold type can be used for topic headings within text. Avoid making random changes in your use of type conventions. Consistently apply the same fonts and the same combination of styles throughout your Web site; consistency develops a strong visual identity. The Web Style Guide Web site (*www.webstyleguide.com*) shown in Figure 5-1 is a good example of effective type usage. The site has a strong typographic identity, yet uses only two typefaces. The designers of this site built a visually interesting page simply by varying the weight, size, white space, and color of the text.

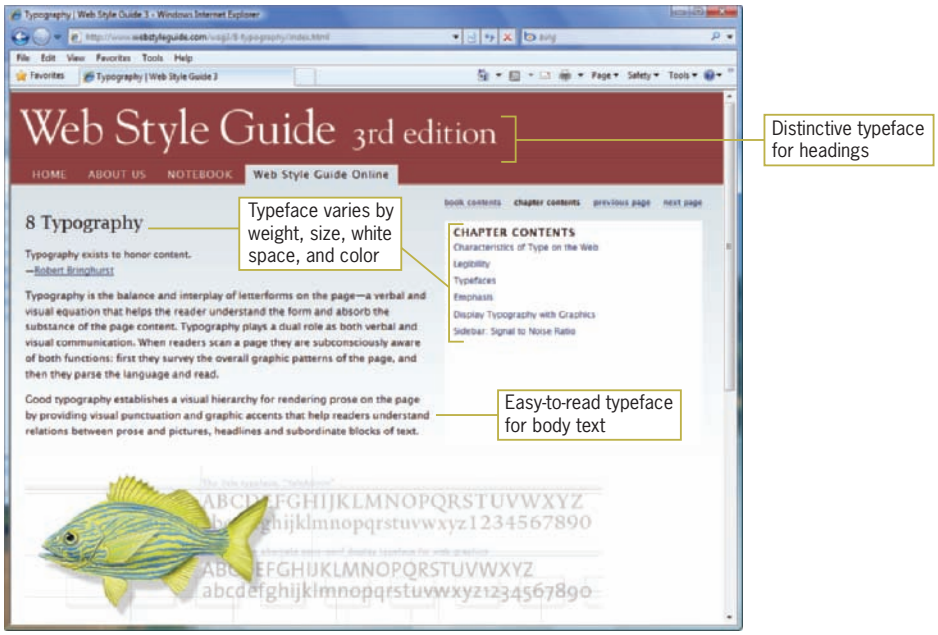


Figure 5-1 Effective typographic design

Use Common Web Fonts

Fonts often are a problem in HTML because font information is client based. The user's browser and operating system determine how a font is displayed, or if it is displayed at all. If you design your pages using a font that your user does not have installed, the browser defaults to Times on a Macintosh or Times New Roman on a PC. To make matters worse, even the most widely available fonts appear in different sizes on different operating systems. Unfortunately, the best you can do about this is to test on multiple platforms to judge the effect on your pages.

To control more effectively how text appears on your pages, think in terms of font families, such as serif and sans-serif typefaces (see Figure 5-2), rather than specific styles. Notice that serif fonts have strokes (or serifs) that finish the top and bottom of the letter. Sans-serif fonts consist of block letters without serifs.

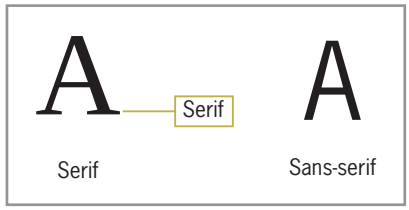


Figure 5-2 Serif and sans-serif type

Because of the variable nature of fonts installed on different computers, you never can be sure the user will see the exact font you have specified. You can, however, specify font fallback values (described later in this chapter), which let you specify a variety of fonts within a font family, such as the common sans-serif fonts, Arial and Helvetica.

Table 5-1 lists the most common installed fonts on the PC, Macintosh, and Linux operating systems according to the Code Style Web site (*www.codestyle.org*).

Common PC Fonts	Common Macintosh Fonts	Common Linux Fonts
Arial	Helvetica	Helvetica
Courier New	Courier	Times
Times New Roman	Times	*URW Chancery L*
Trebuchet MS	Trebuchet MS	Century Schoolbook
Verdana	Verdana	URW Gothic L
Tahoma	Arial	URW Bookman L
Comic Sans MS	Geneva	Nimbus Mono L
Lucida Console	Lucida Grande	URW Palladio L
Georgia	Monaco	DejaVu Sans Mono

Table 5-1 Common Installed Fonts

The fonts that become the most common for the Web are the result of the Core Fonts for the Web initiative, started by Microsoft in 1996. Their font pack, which was freely distributed, contained Arial, Georgia, Verdana, and Times New Roman. As Table 5-1 shows, Times (or Times New Roman) is available on all three operating systems; it is the default browser font. Courier is the default monospace font, and Arial or Helvetica is usually the default sans-serif font. Arial, Trebuchet, and Verdana come with Internet Explorer, so many Macintosh and PC users have these fonts installed. Some Macintosh users only have Helvetica, so it is a good idea to specify this font as an alternate choice when you are using sans-serif fonts.

Specifying Proprietary Web Fonts

Web designers have traditionally been limited to choosing fonts that are installed on users' systems. The CSS3 font-face property lets you link to a font, download it, and it use it in style rules as if it were installed on the user's computer. The common browsers support the font-face property, though they each implement it differently. Internet Explorer uses its proprietary Embedded Open Type format, while the other major browsers use TrueType or Open Type formats. This inconsistency means that if you use the font-face property, you need to specify multiple fonts to satisfy the needs of different browsers. To solve this problem, Microsoft, Mozilla, and Opera have collaborated on a new type format named the Web Open Font Format (WOFF). WOFF was developed during 2009 and is on its way to becoming a Web standard in 2010. WOFF is designed to offer a single interoperable format for Web fonts that all browsers support.

The font-face property opens a new range of fonts to make Web pages more attractive and legible, but in many instances Web designers or the clients they work with must be prepared to pay licensing fees for the fonts they want to use. Commercial Web sites that license fonts include Typekit (*www.typekit.com*) and Typotheque (*www.typotheque.com*). Other sources for fonts include the Open Font Library (*openfontlibrary.org/media*) and Font Squirrel (*www.fontsquirrel.com/fontface*). You will learn how to use the font-face property later in this chapter.

Design for Legibility

Figure 5-3 shows the same paragraph in Times, Trebuchet, Arial, Verdana, and Georgia at the default browser size in Firefox and Internet Explorer. Although these two examples look almost identical (Opera and Safari offer the same results), remember that browser version, operating system, and video capabilities can produce variations in the weight, spacing, and rendering of the font families to individual users.

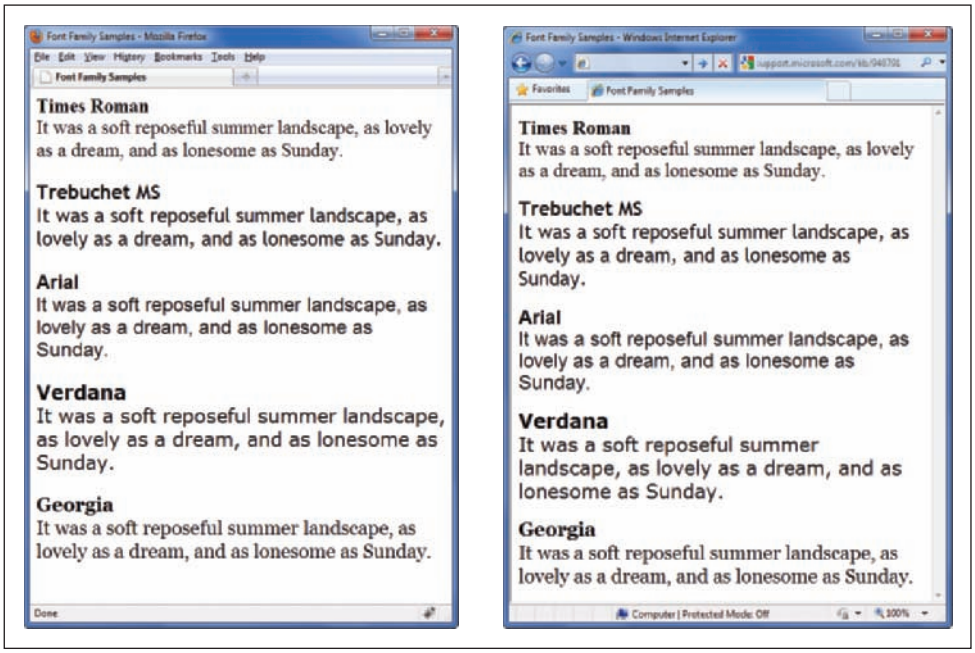


Figure 5-3 Common Web font families in Firefox and Internet Explorer

These examples show that where the text wraps at the end of each line depends on the font and the display characteristics of the browser. Because its **x-height** (the height of the letter *x* in the font) is smaller than that of other fonts, Times can be hard to read, even though it is a serif typeface. This makes it a poor choice for a default font. Trebuchet is a sans-serif face that has a large x-height and rounded letter forms for easy screen legibility. Arial is widely available and is the most commonly used sans-serif font. Verdana is an expanded font—each letter takes up more horizontal space than letters in the other font families. This makes the text easier to read online, but takes much more space on the page.

The size and face of the type you use on your pages determine the legibility of your text. The computer screen has a lower resolution than the printed page, making fonts that are legible on paper more difficult to read on screen. Keep fonts big enough to be legible, and avoid specialty fonts that degrade when viewed online. To aid the reader, consider adding more white space to the page around your blocks of text and between lines as well. Test your content with both serif and sans-serif body text. Finally, make sure that you provide enough contrast between your text color and the background color; in general, darker text on a light background is easiest to read.

Avoid Creating Text as Graphics

The increased number of common fonts and the availability of specialty fonts means that fewer Web designers must resort to creating graphics simply to present text. This technique used to be common in the earlier days of Web design when font choices were more restricted. Still, most Web sites use text graphics in one form or another whether for a main logo, banner, or advertisement. Some sites still use images for navigation graphics, which can be easily substituted with CSS. Because you add download overhead with every additional graphic, save text graphics for important purposes, as described here. Remember that including text as graphics means users cannot search for that text, and that your content will not be accessible to users with screen readers and other adaptive devices. Whenever possible, use HTML-styled text on your pages including creating HTML and CSS-based navigation, which you will learn about in Chapter 9.

You can control how your Web pages look when they are printed with a print style sheet, described in Appendix D.

Understanding CSS Measurement Units

CSS offers a variety of measurement units, including absolute units, such as points, and relative units, such as ems. The measurement values you choose depend on the destination medium for your content. For example, if you are designing a style sheet for printed media, you can use absolute units of measurement, such as points or centimeters. (See Appendix D, Print Style Sheets, for more information.) When you are designing a style sheet for a Web page, you can use relative measurement values that adapt to the user's display type, such as ems or pixels. In this section, you will learn about the CSS measurement units. These units are detailed in Table 5-2.

Unit	Unit Abbreviation	Description
Absolute Units		
Centimeter	cm	Standard metric centimeter
Inch	in	Standard U.S. inch
Millimeter	mm	Standard metric millimeter
Pica	pc	Standard publishing unit equal to 12 points
Point	pt	Standard publishing unit, with 72 points in an inch
Relative Units		
Em	em	The width of the capital *M* in the current font, usually the same as the font size; 1 em is the default font size, 1.5 em is one-and-one-half times the default font size, and so on
Ex	ex	The height of the letter *x* in the current font
Pixel	px	The size of a pixel on the current monitor
Percentage	Example: 150%	Works exactly like em: 100% is the default font size, 150% em is one-and-one-half times the default font size, and so on

Table 5-2 CSS Measurement Units

Absolute Units

CSS lets you use absolute measurement values that specify a fixed value. The measurement values require a number followed by one of the unit abbreviations listed in Table 5-2. By convention, do not include a space between the value and the measurement unit. The numeric value can be a positive value, negative value, or fractional value. For example, the following rule sets margins to 1.25 inches:

```
p {margin: 1.25in;}
```

You generally want to avoid using absolute units for Web pages because they cannot be scaled to an individual user's display type. Absolute units are appropriate when you know the exact measurements of the destination medium. For example, if you know a document will be printed on 8.5 × 11-inch paper, you can plan your style rules accordingly because you know the physical dimensions of the finished document. For this reason, absolute units are better suited to print destinations than Web destinations. Although the point is the standard unit of measurement for type sizes, it is not the best measurement value for the Web. Because computer displays vary widely in size, they lend themselves better to relative units of measurement that can adapt to different monitor sizes and screen resolutions.

Relative Units

The relative units are designed to let you build scalable Web pages that adapt to different display types and sizes. This practice ensures that your type sizes are properly displayed relative to each other or to the default font size set for the browser.

Relative units are always relative to the inherited size of their containing element. For example, the following rule sets the font size for the <body> element to 1.5 times (150%) the size of the browser default:

```
body {font-size: 150%;}
```

Child elements inherit the percentage values of their parents. For example, a <p> element within this body element inherits the 150% sizing.

The em Unit

The em is a printing measurement, traditionally equal to the horizontal length of the capital letter *M* in any given font size. In CSS, the **em unit** is equal to the font size of an element. It can be used for both horizontal and vertical measurement. In addition to stating font sizes, em is useful for padding and margins. You can read more about this in Chapter 6.

The size of the em is equivalent to the font size of the element. For example, if the default paragraph font size is 12-point text, the em equals 12 points. Stating a text size of 2em creates 24-point text—two times the default size. This is useful because it means that measurements stated in em are always relative to their environment. For example, assume that you want a large heading on your page. If you set the <h1> element to 24 points, it always remains that size. If a user sets his or her default font size to 24 points, the headings are the same size as the text. However, if you use the relative em unit, the size of the heading is always based on the size of the default text. The following rule sets heading divisions to twice the size of the default text:

```
div.heading {font-size: 2em;}
```

Percentage

Percentage values for fonts work exactly the same as ems described above. For example, if the default paragraph font size is 12-point text, a 100% font size equals 12 points. A font size set to 125% based on a 12-point default would be 15 points.

The ex Unit

The **ex unit** is equal to the height of the lowercase letter *x* in any given font. As shown in Figure 5-4, the height of the lowercase letter *x* varies widely from one typeface to another.

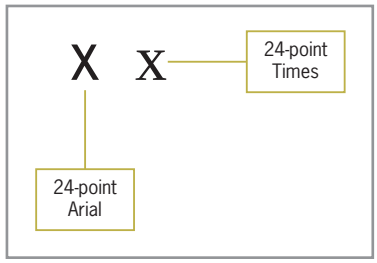


Figure 5-4 Differences in height in the ex unit

Ex is a less reliable unit of measurement than em because the size of the letter *x* changes in height from one font family to another, and the browser cannot always calculate the difference correctly. Most browsers simply set the ex value to one-half the value of the font's em size. Ex is not commonly used to set font sizes.

The px Unit

Pixels are the basic picture element of a computer display. The size of the pixel is determined by the display resolution. Resolution is the measure of how many pixels fit on a screen. As the resolutions grow in value, the individual pixel size gets smaller, making the pixel relative to the individual display settings. Pixel measurements work well for computer displays, but they are not so well suited to other media, such as printing, because some printers cannot accurately determine the size of the pixel. Pixels are not a good choice for font sizes because they do not adapt well when the user changes the size of their display fonts. Ems or percentages are always the best choice.

CSS Property Descriptions

The property descriptions on the following pages and in other chapters provide key information about each CSS property. A property description looks like the following:

border-width property description	
Value:	thin | medium | thick | <length>
Initial:	medium
Applies to:	all elements
Inherited:	no
Percentages:	N/A

Table 5-3 lists the five property description categories.

Category	Definition
Value	The valid keyword or variable values for the property; variable values are set between angle brackets; for example, <length> means enter a length value; (Table 5-4 lists the value notation symbols)
Initial	The initial value of the property
Applies to	The elements to which the property applies
Inherited	Indicates if the property is inherited from its parent element
Percentages	Indicates if percentage values are allowed

Table 5-3 Property Description Categories

Table 5-4 lists the value category notation.

Notation	Definition
< >	Words between angle brackets specify a variable value; for example, <length>
|	A single vertical bar separates two or more alternatives, one of which must occur; for example, thin | medium | thick
||	Two vertical bars separate options; one or more of the values can occur in any order; for example, underline || overline || line-through
[ ]	Square brackets group parts of the property value together; for example, none | [underline || overline || line-through] means that the value is either none or one of the values within the square brackets
?	A question mark indicates that the preceding value or group of values is optional

Table 5-4 Value Category Notation

Using the CSS Font Properties

The CSS font properties allow you to control the appearance of your text. These properties describe how the form of each letter looks. The CSS text properties, covered later in this chapter, describe the spacing around the text rather than affecting the actual text itself. In this chapter, you will learn about the following properties:

- font-family
- font-face
- font-size
- font-style
- font-variant
- font-weight
- font-stretch
- font-size-adjust
- font (shorthand property)

Specifying Font Family

font-family property description	
Value:	<family-name> | <generic-family>
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The font-family property lets you state a generic font-family name, such as sans-serif, or a specific font-family name, such as Helvetica. You can also string together a list of font families, separated by commas, supplying a selection of fonts that the browser can attempt to match. Font names containing more than one word must be quoted.

When considering fonts for your Web designs, start by thinking in terms of font families, such as serif and sans-serif typefaces, rather than specific styles. If you are not using licensed fonts, be aware of the variable nature of fonts installed on different computers. You

can never be sure that the user will see the exact font you have specified. You can, however, use font fallback values to specify a variety of fonts within a font family, such as Arial or Helvetica, which are both common sans-serif fonts.

Generic Font Families

You can use the following generic names for font families:

- **Serif** fonts are the traditional letter form, with strokes (or serifs) that finish off the top and bottom of the letter. The most common serif font is Times.
- **Sans-serif** fonts have no serifs. They are block letters. The most common sans-serif fonts are Helvetica and Arial.
- **Monospace** fonts are fixed-width fonts. Every letter has the same horizontal width. Monospace is commonly used to mimic typewritten text or for programming code. The style rules and HTML code in this book are printed in Courier, a monospace font.
- **Cursive** fonts are designed to resemble handwriting. Although often displayed as Comic Sans, this choice can provide inconsistent results.
- **Fantasy** fonts are primarily decorative. Fantasy is not a widely used choice.

The practice of using generic names ensures greater portability across browsers and operating systems because it does not rely on a specific font being installed on the user's computer. The following rule sets <p> elements to the default sans-serif font:

```
p {font-family: sans-serif;}
```

Of course, if you don't specify any font family, the browser displays the default font, usually some version of Times. Figure 5-5 shows the generic font families in Internet Explorer, Firefox, Opera, and Safari. Notice the difference in the display size of the monospace font. Also notice that the cursive font is actually Comic Sans, except in Internet Explorer 8, where it does not resemble handwriting. If a certain font is not available, a different font will be substituted, based on the user's operating system.

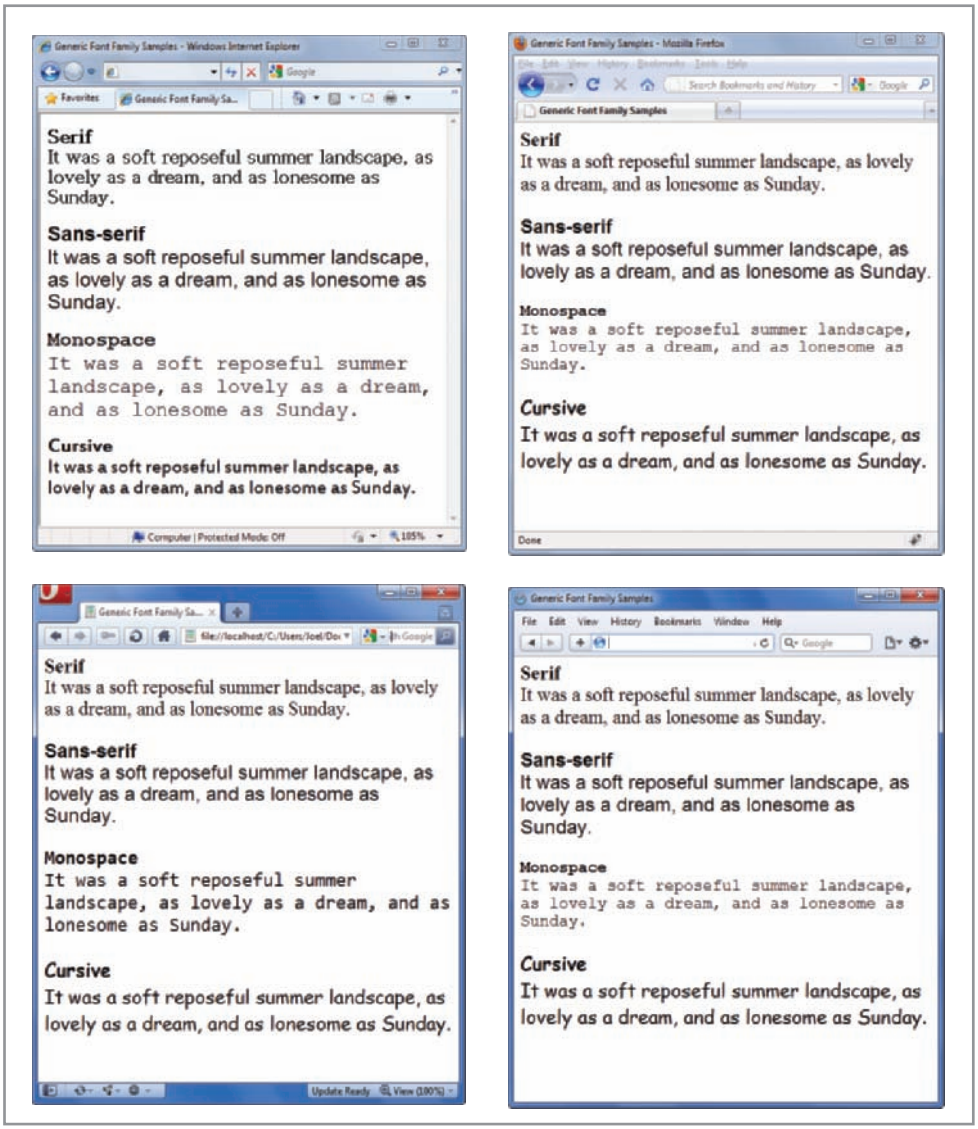


Figure 5-5 Generic font families in Internet Explorer, Firefox, Opera, and Safari

Specific Font Families

In addition to generic font families, the font-family property lets you declare a specific font family, such as Futura or Garamond. The user must have the font installed on his or her computer; otherwise, the browser uses the default font. If the font family name contains white space, such as "lucida console," the font name must be contained within quotes.

The following rule specifies Lucida Console as the font family for the <p> element:

```
p {font-family: "lucida console";}
```

Font Fallbacks

You can specify a list of alternate fonts using a comma as a separator. The browser attempts to load each successive font in the list. If no fonts match, the browser falls back to the default font. The following code tells the browser to use Arial; if Arial is not present, use Helvetica.

```
p {font-family: arial, helvetica;}
```

This font substitution string produces a sans-serif font on PCs that have Arial installed and Macintosh computers that have Helvetica installed. To further ensure the portability of this rule, add a generic font family name to the list, as shown in the following rule:

```
p {font-family: arial, helvetica, sans-serif;}
```

This rule ensures that the <p> element is displayed in some type of sans-serif font, even if it is not Arial or Helvetica.

You should always include a default generic font, such as sans-serif, as the last choice in your font fallback choices. Doing so makes sure that the browser does not use its default font.

Specifying Font-Face

The @font-face property lets you specify a font to be downloaded and displayed in the browser, overcoming the limitations of only using fonts that reside on a user's computer. The font-face property lets you define the name and location of the desired font. Fonts are usually specified in the TrueType Format (TTF). You then specify the font using the font-family property. The following code shows an example of the font-face property.

```
@font-face {font-family: generica;
   src: url(http://www.generic.com/fonts/generica.ttf)}

h1 {font-family: generica, serif;}
```

Remember to always include fallback values in case the browser has a problem downloading the primary font.

The following code shows an example of specifying type in the two formats:

```
@font-face {
   font-family: generic;

/* EOT Format */
   src: url(http://www.generic.com/fonts/generica.eot)}
```

Most modern browsers support the @font-face property, but as described earlier, Internet Explorer 8 and earlier use a proprietary Microsoft font format. This means you must specify two versions of the same font, one in the Microsoft EOT format and one in TTF format. You can convert fonts to Microsoft's format with one of the following online converters. Remember to specify fallback values that fit your design needs.

www.microsoft.com/typography/weft.mspx

www.kirsle.net/wizards/ttf2eot.cgi

Most Web fonts are proprietary and must be purchased and licensed for use. Using licensed fonts protects you from legal liability and also protects the rights of the font designers.

```
/* TTF Format */
   src:
url(http://www.generic.com/fonts/generica.eot)format("truetype");
}
```

Specifying Font Size

font-size property description	
Value:	<absolute-size> | <relative-size> | <length> | <percentage>
Initial:	medium
Applies to:	all elements
Inherited:	computed value is inherited
Percentages:	refer to parent element's font size

The font-size property gives you control over the specific sizing of your type. You can choose from length units, such as ems or pixels, or a percentage value that is based on the parent element's font size.

The following rule sets the <blockquote> element to 1.5 em Arial:

```
blockquote {font-family: arial, sans-serif; font-size: 1.5em;}
```

To specify a default size for a document, use *body* as the selector. This rule sets the text to .85 em Arial:

```
body {font-family: arial, sans-serif; font-size: .85em;}
```

You can also choose from a list of absolute size and relative size keywords, described in the following sections.

Absolute Font Size Keywords

These keywords refer to a table of sizes that is determined by the browser. The keywords are:

- xx-small
- x-small
- small
- medium
- large
- x-large
- xx-large

The CSS specification recommends a scaling factor of 1.2 between sizes for the computer display. Therefore, if the medium font is 10 points, the large font would be 12 points (10 × 1.2 = 12).

Solving the Font Size Dilemma

Figure 5-6 shows a variety of font size samples in the Google Chrome browser. With so many methods of font sizing available, which should you use? The designers of CSS2, Hakon Lie, and Bert Bos, recommend always using relative sizes (specifically, the em value) to set font sizes on your Web pages. Here are some reminders about each relative measurement value:

- *Ems or percentage*—The best choice for fonts because ems and percentages are scalable based on the user's default font size. For the same reason, padding and margins specified in ems are another way to make Web pages more adaptable.
- *Pixels*—Pixel values entirely depend on the user's screen resolution, making it very difficult to ensure consistent presentation. Pixels are a good choice for borders and other design elements, but not a good choice for fonts.

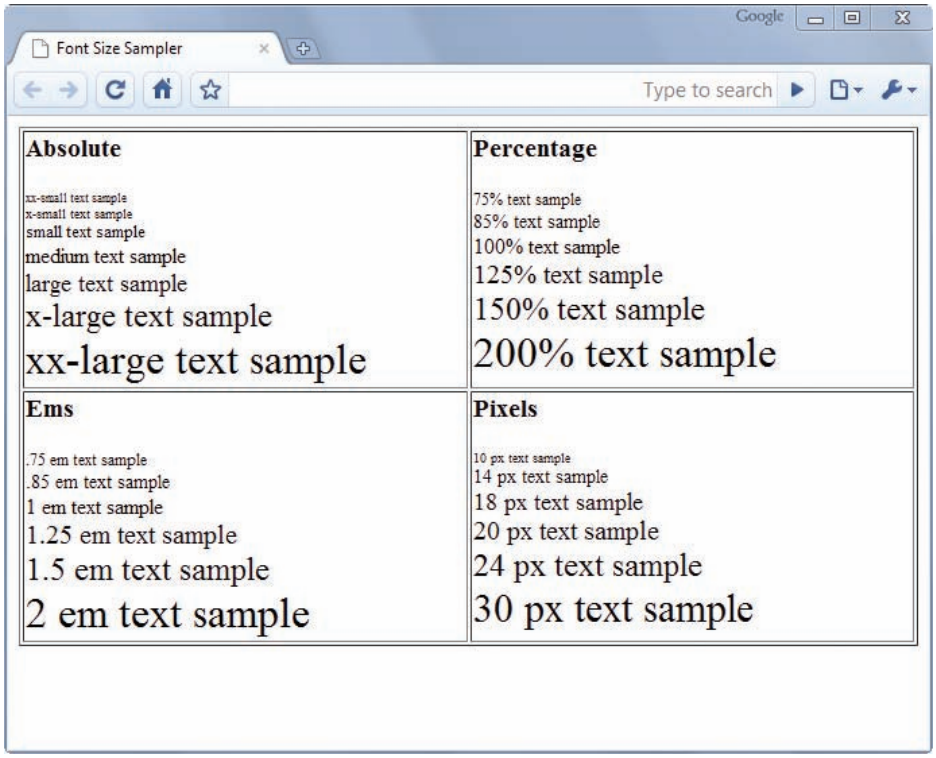


Figure 5-6 Various font sizes

Specifying Font Style

font-style property description	
Value:	normal | italic | oblique
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The font-style property lets you specify italic or oblique text. The difference between italic and oblique text is subtle. The italic form of a typeface is designed with different letter forms to create the slanted font, while the oblique form is simply normal text slanted to the right. In print-based typography, oblique text is considered inferior to italic. On the Web, however, current browsers cannot make the distinction between the two—either value creates slanted text. The following example sets italicized text for the note class attribute.

```
.note {font-style: italic;}
```

Here is the note class applied to a element:

```
<div class="note">A note to the reader:</div>
```

The text contained in the appears italicized in the browser. Remember that italic text is hard to read on a computer display. Use italics for special emphasis rather than for large blocks of text.

Specifying Font Variant

font-variant property description	
Value:	normal | small-caps
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The font-variant property lets you define small capitals, which are often used for chapter openings, acronyms, and other special purposes. Small capitals are intended to be a different type style from regular capital letters, but this is not supported in all browsers. In fact, some simply downsize the regular capital letters. Figure 5-7 shows an example of small capitals.

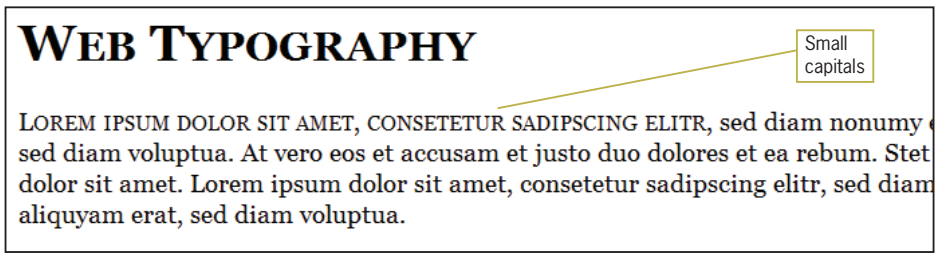


Figure 5-7 Small capitals add a distinctive look

In this example, a style rule specifies a class named *small* to be used with a element. The element in the HTML code contains the text that is converted to small capitals.

```
span.small {font-variant: small-caps}
```

```
<p><span class="small">I first heard of Antonia</span> on what
seemed to be...</p>
```

Specifying Font Weight

font-weight property description	
Value:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The font-weight property lets you set the weight of the typeface. The numeric values express nine levels of weight from 100 to 900, although most browsers and fonts do not support such a wide range of weights. The default type weight is equal to 400, with bold text equal to 700. Bolder and lighter are relative weights based on the weight of the parent element. Using the bold value produces the same weight of text as the element. The following style rule sets the *warning* class to bold:

```
.warning {font-weight: bold;}
```

Specifying Font Stretch

font-stretch property description	
Value:	normal | wider | narrower | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The font-stretch property is a CSS3 property that currently has limited support in browsers.

The font-stretch property lets you expand or compress the font face. Not all forms of the font may be available, so the next closest choice is substituted: the closest condensed face substitutes for any condensed choice, and the closest expanded face for any expanded choice. The following rule expands the h1 font face if available.

```
h1 {font-family: sans-serif; font-stretch: expanded}
```

Using the Font Shortcut Property

font property description	
Value:	[[<'font-style'> || <'font-variant'> || <'font-weight'>]? <'font-size'> [/ <'line-height'>]? <'font-family'>]
Initial:	see individual properties
Applies to:	all elements
Inherited:	yes
Percentages:	allowed on 'font-size' and 'line-height'

The **font property** is a shortcut that lets you specify the most common font properties in a single statement. The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts.

As shown in the value listing for the font shortcut property, the font property lets you state the font-style, font-variant, font-weight, font-size, line-height, and font-family in one statement. The only two values that are required are font-size and font-family, which must be in the correct order for the style rule to work. The following rules are examples of basic use of the font property:

```
p {font: 12pt arial, sans-serif;}
h1 {font: 2em sans-serif;}
```

The font properties other than font-size and font-family are optional and do not have to be included unless you want to change

their default. If you want to include line-height, note that it must always follow a slash after the font-size. The following rule sets .85em Arial text on 1em line height:

```
p {font: .85em/1em arial;}
```

The font shortcut property lets you abbreviate the more verbose individual property listings. For example, both of the following rules produce the same result:

```
p {font-weight: bold;
   font-size: .85em;
   line-height: 1em;
   font-family: arial;
}

p {font: bold .85em/1em arial;} /* Same rule as above */
```

Although the font shortcut property is a convenience, you may prefer to state explicitly the font properties as shown in the more verbose rule, because they are easier to understand. It is also a good idea to choose a convention of using either the individual property names or the shortcut notation, and then use the convention consistently throughout your Web site.

Using the CSS Text Spacing Properties

The CSS text properties let you adjust the spacing around and within your text and add text decorations. The properties in this section let you create distinctive text effects. In this section, you will learn about the following properties:

- text-indent
- text-align
- line-height
- vertical-align
- letter-spacing
- word-spacing
- text-decoration
- text-transform
- text-shadow

Specifying Text Indents

text-indent property description	
Value:	<length> | <percentage>
Initial:	0
Applies to:	block-level element
Inherited:	yes
Percentages:	refer to width of containing block

Use the text-indent property to set the amount of indentation for the first line of text in an element, such as a paragraph. You can specify a length or percentage value. The percentage is relative to the width of the containing element. If you specify a value of 15%, the indent will be 15% of the width of the element. Negative values let you create a hanging indent. Figure 5-8 shows two text-indent effects.

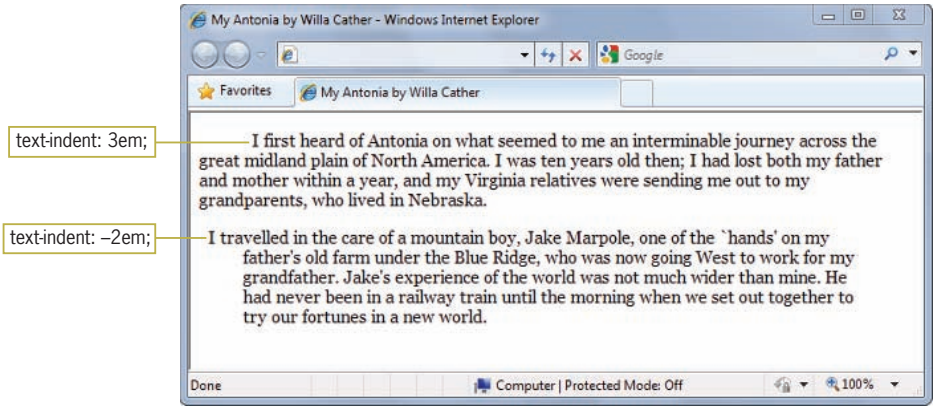


Figure 5-8 Text indents

The following rules set an indent of 2em for the <p> element and -2em for the <blockquote> element:

```
p {text-indent: 2em;}
blockquote {text-indent: –2em;}
```

Indents are sensitive to the language specification for the document. You can specify the document language with the *lang* attribute in the opening <html> tag, such as <html lang="en">. In left-to-right reading languages (such as English), the indent is added to the left of the first line; in right-to-left reading languages (such as Hebrew), the indent is added to the right of the first line.

Indents are inherited from parent to child elements. For example, the following rule sets a 2em text indent to a element:

```
div {text-indent: 2em;}
```

Any block-level elements, such as <p>, that are contained within this division have the same 2em text indent specified in the rule for the parent .

You can find a complete list of the standard language codes here: *www.loc.gov/standards/iso639-2/php/code_list.php*

Specifying Text Alignment

text-align property description	
Value:	left | right | center | justify
Initial:	depends on user agent and language
Applies to:	block-level elements
Inherited:	yes
Percentages:	N/A

Use the text-align property to set horizontal alignment for the lines of text in an element. You can specify four alignment values: left, center, right, and justify. The justify value lines up the text on both horizontal margins, adding white space between the words on the line, like a column of text in a newspaper. The following style rule sets the <p> element to justified alignment:

```
p {text-align: justify;}
```

Figure 5-9 shows a sample of all four alignment values.

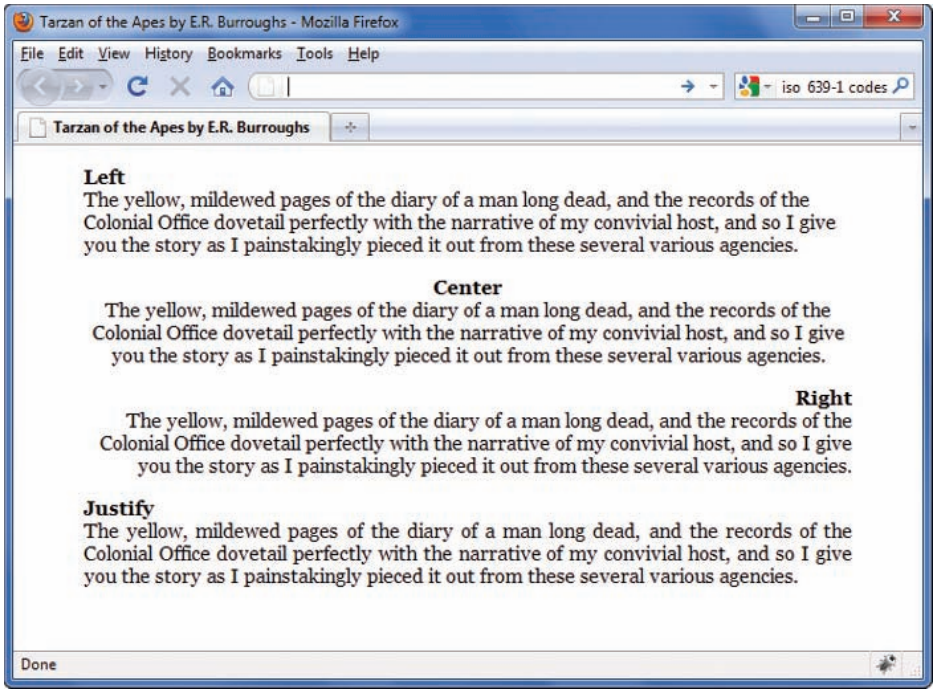


Figure 5-9 Text alignments

When choosing an alignment value, keep the default settings for the language and the user's preferences in mind. For example, most Western languages read from left to right, and the default alignment is left. Unless you are trying to emphasize a particular section of text, use the alignment with which most readers are comfortable. Both right and center alignment are fine for short sections of text, but they make reading difficult for lengthier passages.

Justified text lets you create newspaper-like alignment where the lines of text all have the same length. The browser inserts white space between the words of the text to adjust the alignment so both margins of the text align, as shown in Figure 5-10. Justify is not supported by all browsers, and different browsers might justify the text differently.

Specifying Line Height

line-height property description	
Value:	normal | <number> | <length> | <percentage>
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	refer to the font size of the element itself

CSS allows you to specify either a length or percentage value for the line height, which is also known as **leading**, the white space between lines of text. The percentage is based on the font size. Setting the value to 150% with a 1em font size results in a line height of 1.5em. The following rule sets the line height to 150%:

```
p {line-height: 150%;}
```

Figure 5-10 shows the default line height and various adjustments in line height. Notice that the line height is evenly divided between the top and bottom of the element.

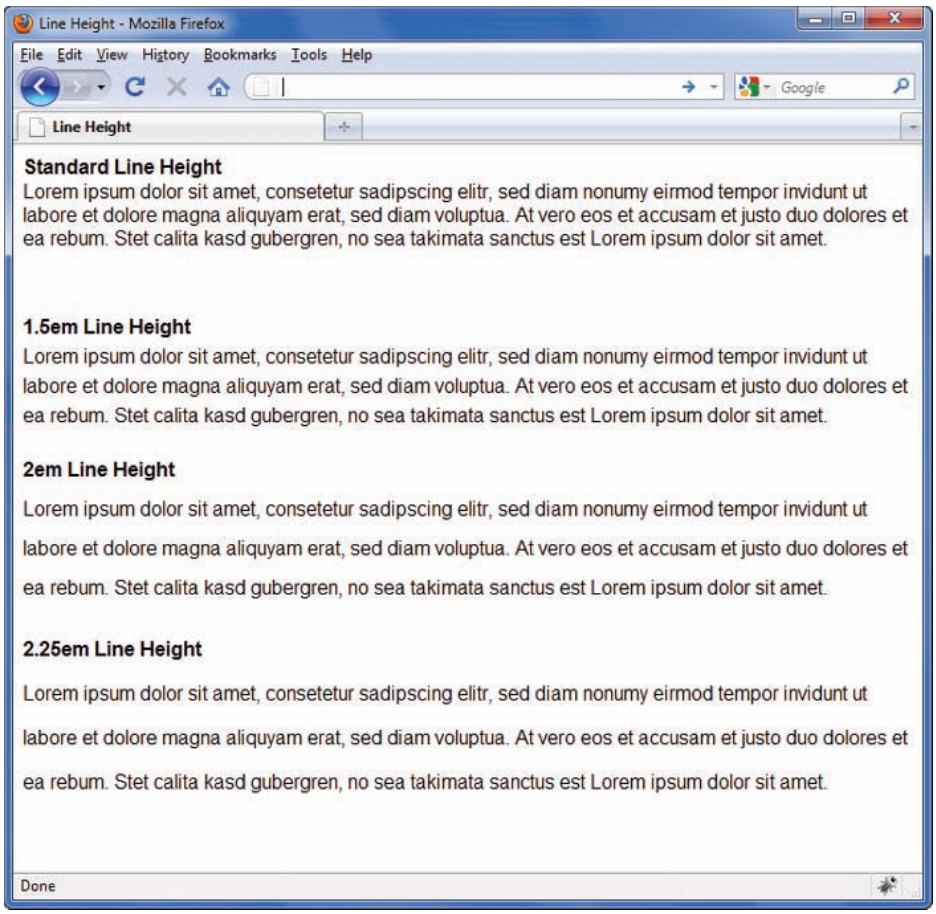


Figure 5-10 Adjusting line height increases legibility

The line-height property can increase the legibility of your text. Adding to the default line height inserts additional white space between the lines of text. On a display device, increasing the white space helps guide the user's eyes along the line of text and provides rest for the eye. As Figure 5-10 shows, increasing the line height adds to the legibility of the text.

Specifying Vertical Alignment

vertical-align property description	
Value:	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length>
Initial:	baseline
Applies to:	inline-level and 'table-cell' elements
Inherited:	no
Percentages:	refer to the 'line-height' of the element itself

The vertical-align property lets you adjust the vertical alignment of text within the line box. Vertical-align works only on inline elements. You can use this property to superscript or subscript characters above or below the line of text and to align images with text. Table 5-5 defines the different vertical-align values. The baseline, sub, and super values are the most evenly supported by the different browsers.

Value	Definition
baseline	Align the baseline of the text with the baseline of the parent element
sub	Lower the baseline of the box to the proper position for subscripts of the parent's box; this value does not automatically create a smaller font size for the subscripted text
middle	The CSS2 specification defines "middle" as "the vertical midpoint of the box with the baseline of the parent box plus half the x-height of the parent"; realistically, this means the middle-aligned text is aligned to half the height of the lowercase letters
super	Raise the baseline of the box to the proper position for superscripts of the parent's box; this value does not automatically create a smaller font size for the superscripted text
text-top	Align the top of the box with the top of the parent element's font
text-bottom	Align the bottom of the box with the bottom of the parent element's font
top	Align the top of the box with the top of the line box
bottom	Align the bottom of the box with the bottom of the line box

Table 5-5 Vertical-align Property Values

The following rule sets superscripting for the superscript class:

```
.superscript {vertical-align: super;}
```

Figure 5-11 shows different types of vertical alignments.

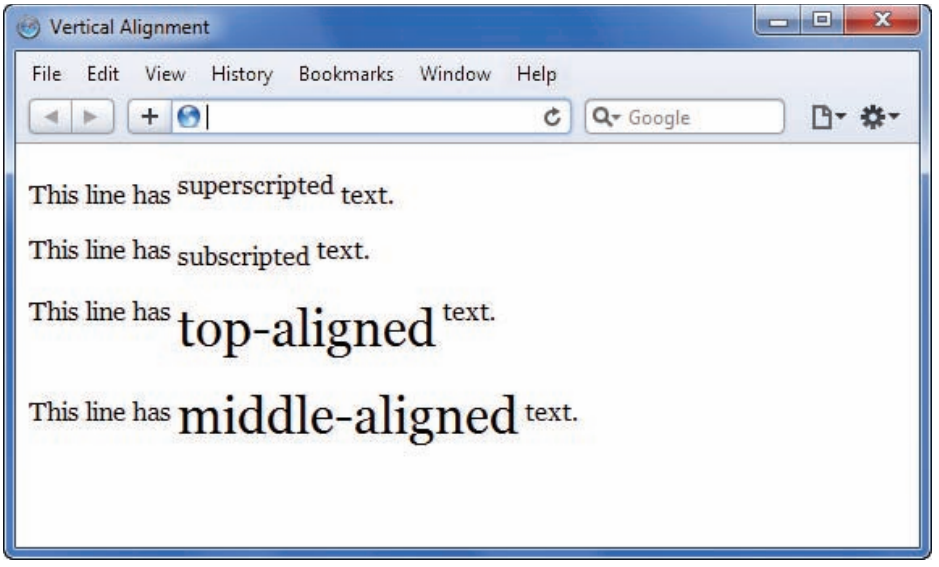


Figure 5-11 Vertical alignments

You can also use vertical alignment to align text with graphics. The following rule, added to the <img> element with the style attribute, sets the vertical alignment to top:

```
<img src="image.gif" style="vertical-align: text-top;">
```

Figure 5-12 shows various alignments of images and text. Note that the vertical alignment affects only the one line of text that contains the graphic, because the graphic is an inline element. If you want to wrap a paragraph of text around an image, use the float property, described in Chapter 6.

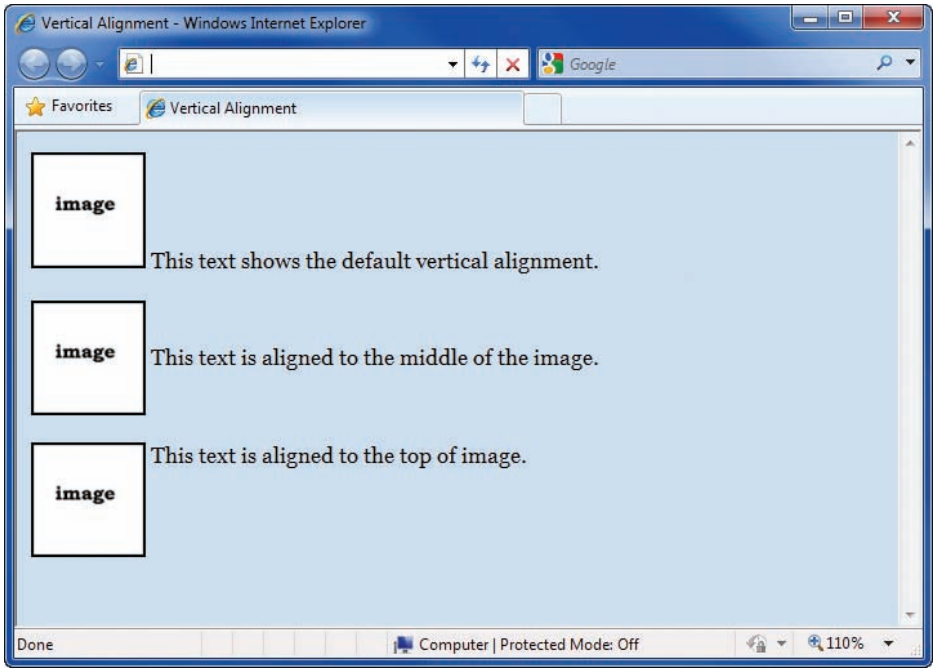


Figure 5-12 Vertically aligning text and graphics

Specifying Letter Spacing

letter-spacing property description	
Value:	normal | <length>
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The letter-spacing property lets you adjust the white space between letters. In publishing terminology, this adjustment is called kerning. The length you specify in the style rule is added to the default letter spacing. The following code sets the letter spacing to 4 pixels:

```
h1 {letter-spacing: 4px;}
```

Figure 5-13 shows samples of different letter-spacing values. The letter-spacing property is an excellent method of differentiating headings from the rest of your text.

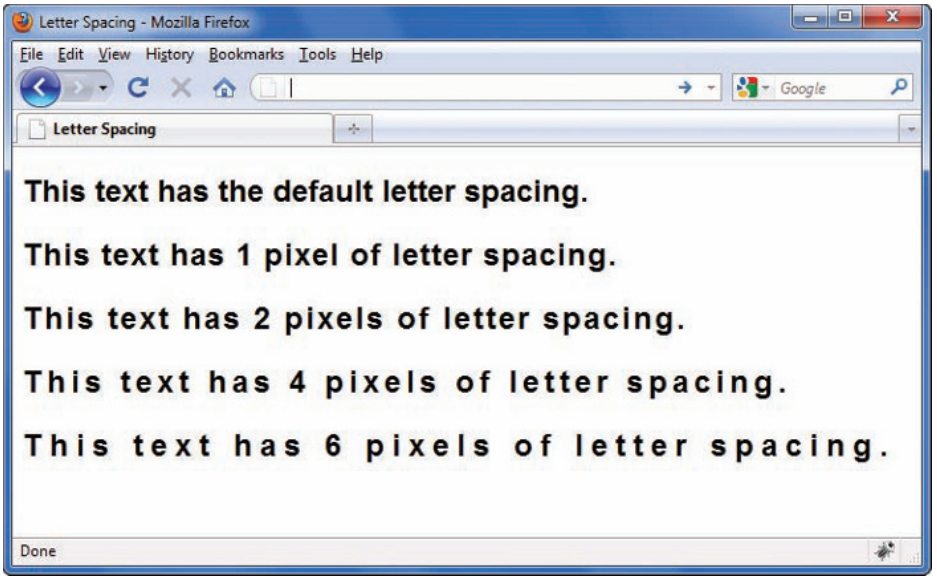


Figure 5-13 Adjusting letter spacing

Specifying Word Spacing

word-spacing property description	
Value:	normal | <length>
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The word-spacing property lets you adjust the white space between words in the text. The length you specify in the style rule is added to the default word spacing. The following code sets the word spacing to 2em:

```
h1 {word-spacing: 2em;}
```

Figure 5-14 shows the result of the word-spacing property. Like the letter-spacing property, word-spacing is an effective way to make your headings stand out.

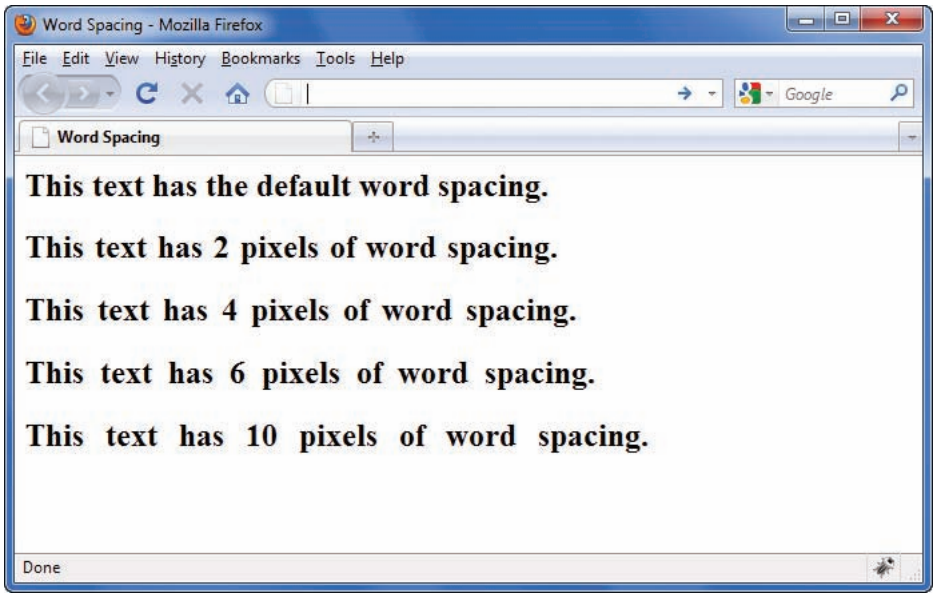


Figure 5-14 Adjusting word spacing

Specifying Text Decoration

text-decoration property description	
Value:	none | [underline || overline || line-through || blink]
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A

Text decoration lets you underline text, an effect that has particular meaning in a hypertext environment. See Figure 5-15. Your users know to look for underlined words as the indicators for hypertext links. Any text you underline appears to be a hypertext link. Except for text links, underlining is an inappropriate text style for a Web page.

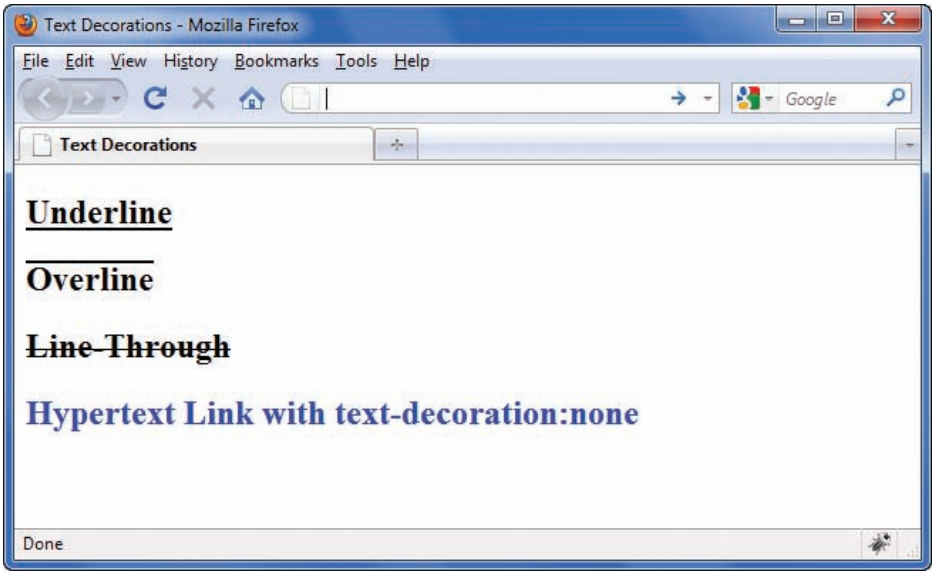


Figure 5-15 Text decorations

As Figure 5-15 shows, the text-decoration property lets you remove the underlining from the <a> element. As you read earlier, the user commonly relies on the underlining text to indicate a hypertext link. However, some Web sites choose to remove link underlining, indicating links with a color different from the standard text color. You can remove the underlining from your anchor elements with the following rule:

```
a {text-decoration: none;}
```

Users with sight disabilities can have trouble finding the links in your content if you choose to remove the underlining. Alternately, a user can override the author's style rules by setting preferences in his or her browser or applying his or her own style sheet.

Specifying Capitalization

The text-transform property lets you change the capitalization of text. This property is very useful for headings anywhere you want to change the capitalization of text from its original capitalization format to a different format without actually editing the text.

text-transform property description	
Value:	capitalize | uppercase | lowercase | none
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The *capitalize* value capitalizes the first letter of every word. *Uppercase* and *lowercase* transform the case of an entire word. The following code transforms the case of an <h1> element to uppercase.

```
h1 {text-transform: uppercase;}
```

Specifying Text Shadow

text-shadow property description	
Value:	none | [<shadow>,] * <shadow>
Initial:	none
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The text-shadow property lets you define a shadow that is displayed behind text. You can specify the vertical and horizontal offset as well as the blur of the shadow. This property is ideal for adding depth and character to headings and other important typographic elements on your Web page, although it is best used sparingly. The following code shows the text-shadow property syntax.

```
h1 {text-shadow: 2px 2px 2px #666;}
```

The first two length values indicate the horizontal and vertical offset from direct alignment with the text. Positive values move to the right and down, negative values move to the left and up.

The third length value specifies the blur amount, which determines how soft the edges of the shadow will display. The final value sets the color of the shadow. Figure 5-16 shows examples of different shadow values both with and without blur.

Remember to test the text-shadow property carefully; although if text-shadow is not supported, the result is not too bad, as the text is simply displayed without a shadow.

The text-shadow property is not supported by Internet Explorer 8.

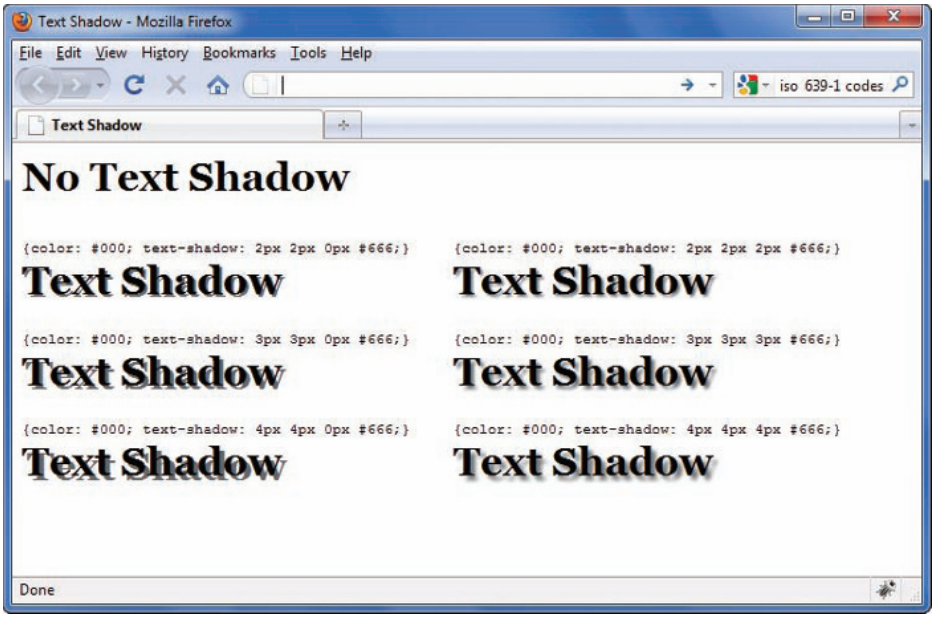


Figure 5-16 Text shadow

Currently Unsupported CSS3 Properties

The CSS3 recommendation contains some properties that are not supported evenly or at all by different browsers and devices. These properties may change, so check the latest CSS3 documentation at *www.w3.org* for the latest updates. As new versions of the browsers are released, these properties should gradually become supported. Table 5-6 contains descriptions of these properties.

Property	Characteristics
white-space Controls how paragraph text wraps, and whether to preserve white space Normally the browser ignores extra spaces between words; you can preserve this space using the *pre* value	Value: normal | pre | nowrap | pre-wrap | pre-line Initial: not defined for shorthand properties Applies to: all elements Inherited: yes Percentages: N/A
text-wrap Controls text wrapping	Value: normal | unrestricted | none | suppress Initial: normal Applies to: all elements Inherited: yes Percentages: N/A

Table 5-6 Currently Unsupported CSS3 Properties *(continues)*

(continued)

Property	Characteristics
word-wrap Controls whether words can be broken to wrap a sentence	Value: normal | break-word Initial: normal Applies to: all elements Inherited: yes Percentages: N/A
text-align-last When text is set to justify, the last line in a paragraph may align unevenly; this property controls the alignment of the last line	Value: start | end | left | right | center | justify Initial: start Applies to: all elements Inherited: yes Percentages: N/A
text-emphasis Intended for Asian languages; adds accent marks above characters for emphasis	Value: none | [[accent | dot | circle | disc] [before | after]?] Initial: none Applies to: all elements Inherited: yes Percentages: N/A
text-outline Specifies a text outline's thickness and blur length	Value: none | [<color> <length> <length>? | <length> <length>? <color>] Initial: none Applies to: all elements Inherited: yes Percentages: N/A
font-stretch The font-stretch property lets you expand or compress the font face; not all forms of the font may be available, so the next closest choice is substituted: the closest condensed face substitutes for any condensed choice, and the closest expanded face for any expanded choice	Value: normal | wider | narrower | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded | inherit Initial: none Applies to: all elements Inherited: yes Percentages: N/A
font-size-adjust For font sizes, the size of the actual characters varies by font—if you are using font substitution, one of the substituted fonts may be smaller and less legible than the primary font you specified; the font-size-adjust property lets you maintain the legibility of your text when substitution occurs. It affects the x-height of the font	Value: <number> | none | inherit Initial: see individual properties Applies to: all elements Inherited: yes Percentages: N/A

Table 5-6 Currently Unsupported CSS3 Properties

Activity: Building a Font and Text Properties Style Sheet

In the following set of steps, you will build a style sheet that uses the typographic techniques you learned about in this chapter. Save your file, and test your work in the browser as you complete each step.

To build the style sheet:

1. Copy the **font activity.html** file from the Chapter05 folder provided with your Data Files to the Chapter05 folder in your work folder. (Create the Chapter05 folder, if necessary.)
2. Open the file **font activity.html** in your HTML editor, and save it in your Chapter05 folder as **font activity1.html**.
3. In your browser, open the file **font activity1.html**. When you open the file, it looks like Figure 5-17.

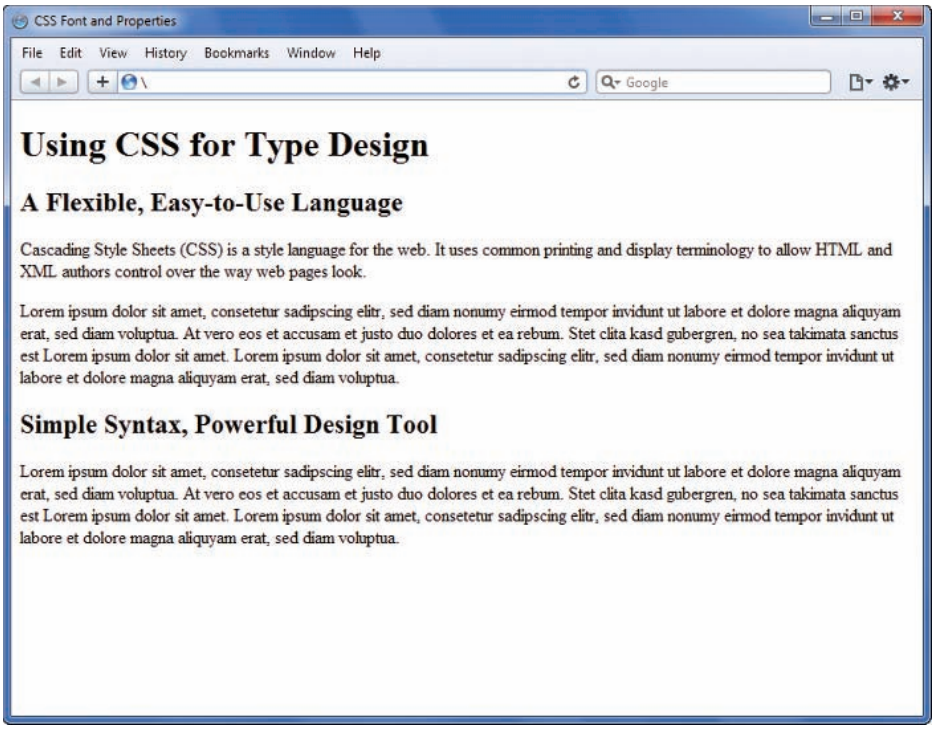


Figure 5-17 Basic HTML document

Adding the <style> Section

Because you are working on a single document, you can use a <style> element in the <head> section to contain your style rules. In the steps throughout the chapter, enter the code shown in bold and blue.

To add the <style> section:

1. Using your text editor, add a <style> element in the <head> section to contain your style rules as shown in the following code. Leave a few lines of white space between the <style> tags to contain the style rules.

```
<head>
<title>CSS Font Activity</title>
<style type="text/css">



</style>
</head>
```

2. Save the file.

Styling the Headings

To style the headings:

1. Write a style rule that selects the <h1> element. Set the font-size to 3em and the font-family to Georgia, with a fallback generic serif font, as shown in the following code:

```
h1 {
   font-size: 3em;
   font-family: georgia, serif;
}
```

2. Write a style rule that selects the <h2>. Set the font-size to 1.5em and the font-family to arial. Add fallback values of helvetica and sans-serif to the font-family declaration.

```
h2 {
   font-size: 1.5em;
   font-family: arial, helvetica, sans-serif;
}
```

3. Save your file and check your work in the browser. Figure 5-18 shows the changes from the style rules you added.

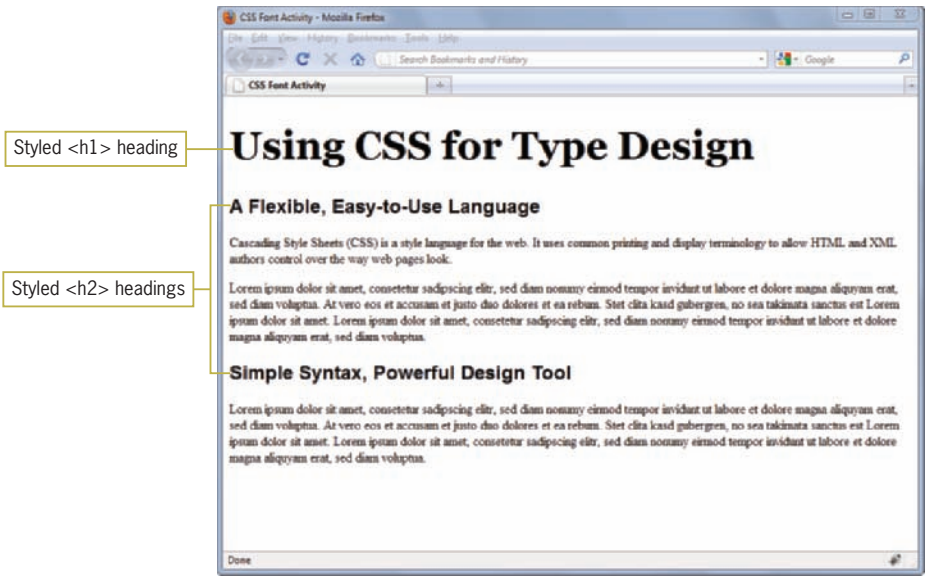


Figure 5-18 Styled headings

Styling the Paragraphs

Write a rule that makes the paragraph text easier to read by changing the font, increasing the white space between lines, and adding a left margin to move the paragraphs away from the edge of the browser window. This rule will also use a class name to apply the style rule to the paragraphs.

To style the paragraphs:

1. Write a style rule that uses a class selector and a class named *copy*. Set the font-family property to *georgia, serif* as you did for the <h1> element. Set the line-height property to 1.5em, and add a margin-left of 20 pixels (20px). You will learn more about margins in Chapter 6.

```
.copy {
   font-family: georgia, serif;
   line-height: 1.5em;
   margin-left: 20px;
}
```

2. Locate the first <p> element within the file and add class="copy" to the opening <p> tag as shown:

```
<p class="copy">Cascading Style Sheets (CSS) is a
style language for the web. It uses common printing
and display terminology to allow HTML and XML authors
control over the way web pages look.</p>
```

3. Repeat Step 2 and add the *class* attribute to the remaining <p> elements. There are three in all.
4. Save your file and check your work in the browser. Figure 5-19 shows the changes from the style rules you added.

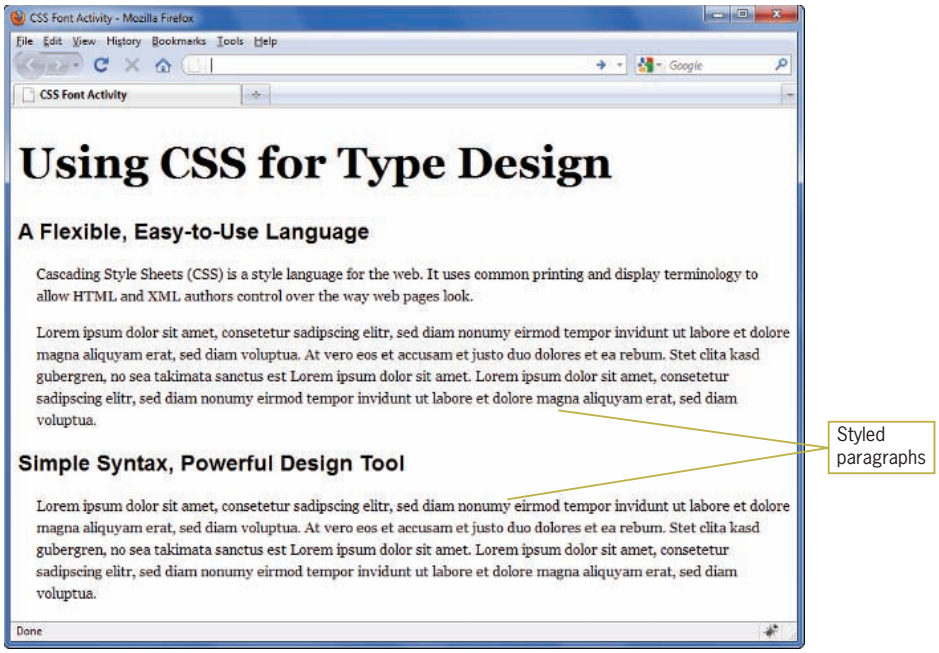


Figure 5-19 Styled paragraphs

Making the Second-Level Headings More Distinctive

The page is looking better with more legible text, but the second-level headings could stand out more. To do this, you will add a left margin and letter-spacing to the existing style rule.

To add styles to the <h2> elements:

1. Include a style rule in the existing h2 selector to add a left margin of 20 pixels to align the headings with the paragraphs.

```
h2 {
   font-size: 1.5em;
   font-family: arial, helvetica, sans-serif;
   margin-left: 20px;
}
```

2. Add one more style rule to set the letter-spacing to 4px.

```
h2 {
   font-size: 1.5em;
   font-family: arial, helvetica, sans-serif;
   margin-left: 20px;
   letter-spacing: 4px;
}
```

3. Save the file and view your changes in the browser. It should look similar to Figure 5-20.

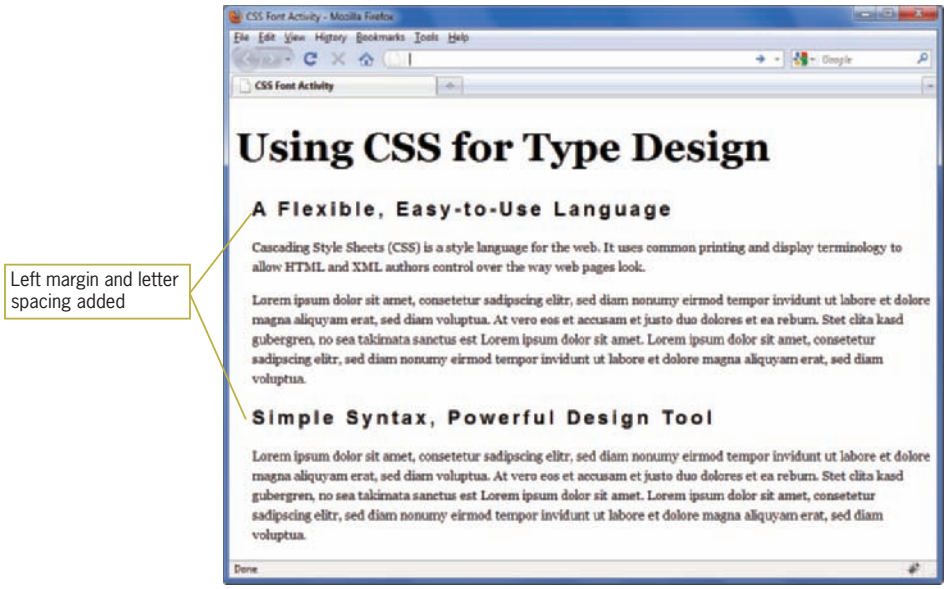


Figure 5-20 New rules added to the <h2> elements

Capitalizing Key Words

You can use the text-transform property along with a element and class selector to select the first few words in the first paragraph and capitalize them, a common printing technique that adds a professional look to the page.

To select and capitalize the words:

1. Write a style rule that uses a class selector for a class named *caps*. Add the text-transform property set to uppercase.

```
.caps {text-transform: uppercase;}
```

2. Add another declaration to the rule, specifying a bold font-weight.

```
.caps {
   text-transform: uppercase;
   font-weight: bold;
}
```

3. Locate the text "Cascading Style Sheets (CSS)" in the first paragraph, and place span tags before and after this text.

```
<p class="copy"><span>Cascading Style Sheets (CSS)
</span> is a style language for the web. It uses common
printing and display terminology to allow HTML and XML
authors control over the way web pages look.</p>
```

4. Add the class attribute to the opening tag and set the value to *caps*.

```
<p class="copy"><span class="caps">Cascading Style
Sheets (CSS)</span> is a style language for the web.
It uses common printing and display terminology to
allow HTML and XML authors control over the way web
pages look.</p>
```

5. Save the file and view your changes in the browser. Figure 5-21 shows the result of the style rule.

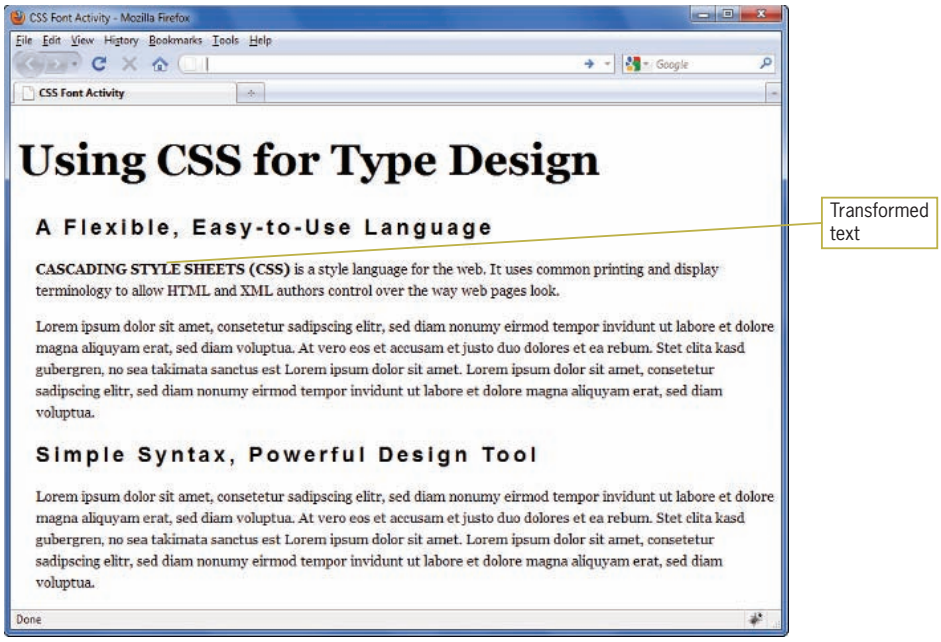


Figure 5-21 Transforming text to uppercase

Customizing Bulleted and Numbered Lists

The list-style properties let you control the visual characteristics of elements that have a display property value of *list-item*, which are the numbered and bulleted lists signified by the <ol> and <ul> elements. These properties let you set the appearance of the marker that indicates each item within the list. With CSS, the marker can be a symbol, number, or image. You can also determine the position of the marker next to the list content.

The two common elements that have a default display value of *list-item* are <ul> and <ol>, which generate a bulleted (unordered) and ordered list, respectively. The following code shows a sample of each type of list:

```
<!-- Bulleted List -->
<h3>Things to do...</h3>
  <ul>
   <li>Buy dog food</li>
   <li>Clean up the house</li>
   <li>Take a rest</li>
  </ul>
<!-- Ordered List -->
<h3>Places to go...</h3>
  <ol>
   <li>Paris</li>
   <li>Sydney</li>
   <li>Cairo</li>
  </ol>
```

Figure 5-22 shows the result of this code. Notice that the default markers are a solid bullet, called a "disc" for the <ul> list, and an Arabic numeral called "decimal" for the <ol> list. You can change these marker values with the CSS list-style properties.

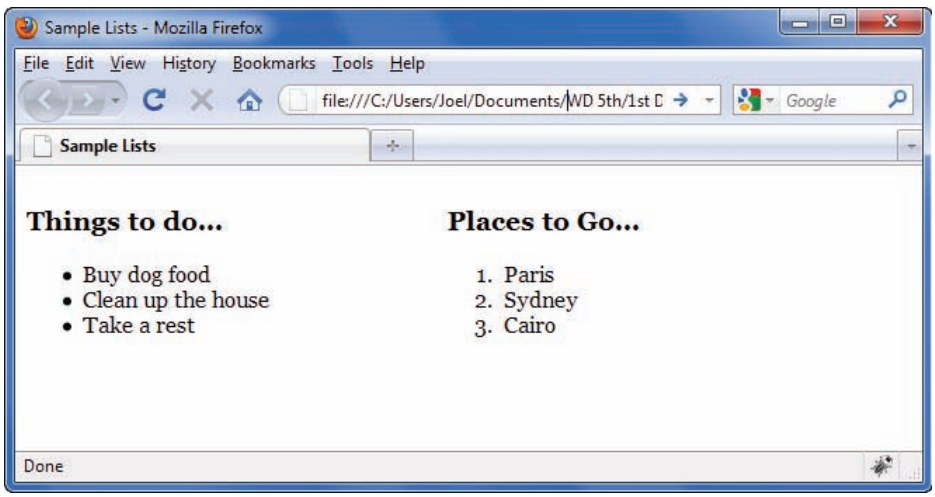


Figure 5-22 Unordered and ordered list elements

Specifying the list-style-type Property

The list-style-type property lets you customize the list marker to a variety of different values.

list-style-type	
Value:	disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-greek | lower-alpha | lower-latin | upper-alpha | upper-latin | hebrew | armenian | georgian | cjk-ideographic hiragana | katakana | hiragana-iroha | katakana-iroha | none
Initial:	disc
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A

The list-style-type property lets you specify one of three types of markers for a list. You can choose a symbol, a numbering system, or an alphabetical system. CSS allows a wide variety of marker values. Remember to test support for these markers in different browsers. Tables 5-7, 5-8, and 5-9 list the different values and their descriptions.

Value	Description
disc	Filled circle (see Figure 5-23)
circle	Hollow circle (see Figure 5-23)
square	Filled square (see Figure 5-23)

Table 5-7 Bulleted List Values

Value	Description
decimal	Decimal numbers, beginning with 1; this is the default numbering
decimal-leading-zero	Decimal numbers padded by initial zeros (01, 02, 03...)
lower-roman	Lowercase roman numerals (i, ii, iii...)
upper-roman	Uppercase roman numerals (I, II, III...)
hebrew	Traditional Hebrew numbering
georgian	Traditional Georgian numbering
armenian	Traditional Armenian numbering
cjk-ideographic	Plain ideographic numbers
hiragana	Japanese hiragana language characters
katakana	Japanese katakana language characters
hiragana-iroha	Japanese hiragana-iroha language characters
katakana-iroha	Japanese katakana-iroha language characters

Table 5-8 Numerical List Values

Value	Description
lower-alpha	Lowercase ASCII letters (a, b, c, ... z)
upper-alpha	Uppercase ASCII letters (A, B, C, ... Z)
lower-greek	Lowercase classical Greek

Table 5-9 Alphabetical list values

Figure 5-23 shows the list types that most browsers support.

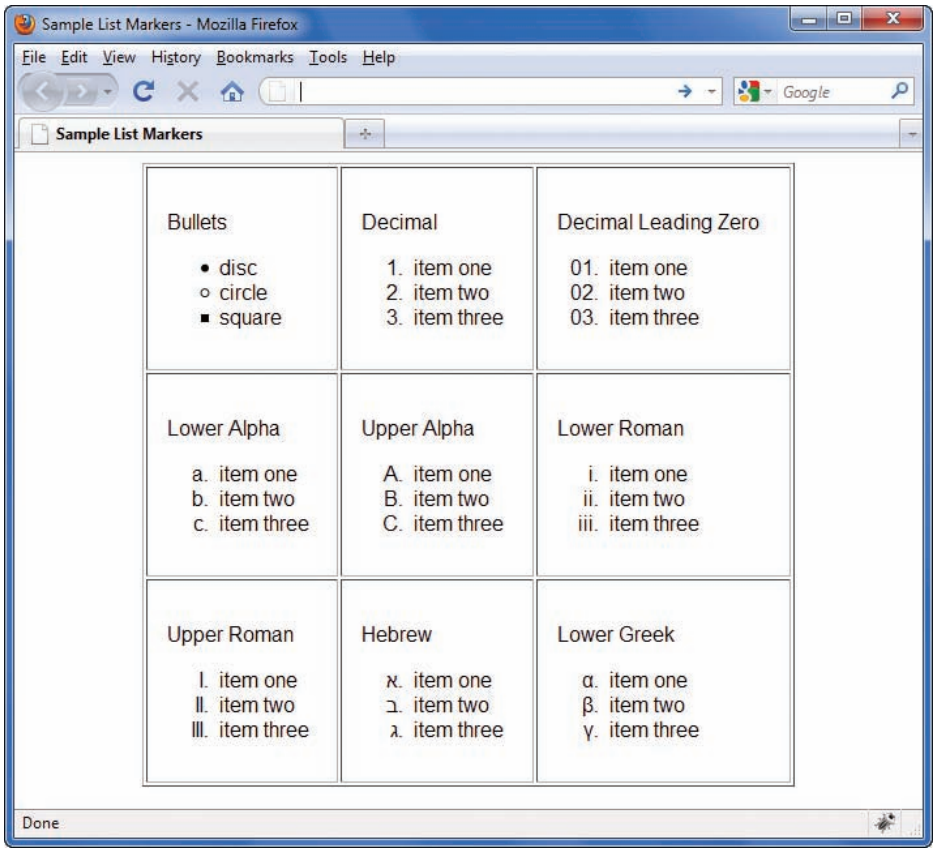


Figure 5-23 Different list marker types

To specify a list-style marker type, select the list container element, either ul or ol, and specify the value as shown in the following style rules:

```
ol {list-style-type: decimal-leading-zero;}
ul {list-style-type: circle;}
```

There may be times when you want to specify a list-style-type within an individual list. You can do this by using the *style* attribute within the <ul> or <ol> element as shown in the following:

```
<ol style="list-style-type: lower-alpha;">
<li>Item One</li>
<li>Item Two</li>
<li>Item Three</li>
</ol>
```

This style rule affects only this one instance of the list.

Specifying the list-style-image Property

The list-style-image property lets you easily attach an image to a list and have it repeated as the marker symbol.

list-style-image	
Value:	<url> | none | inherit
Initial:	none
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual

The list-style-image property lets you replace the standard symbol with an image of your choice. The following code shows the style rule that attaches an image to a bulleted list:

```
ul {list-style-image: url(pawprint.gif);}
```

Figure 5-24 shows the result of the style rule. The image is repeated whenever a list element is used.

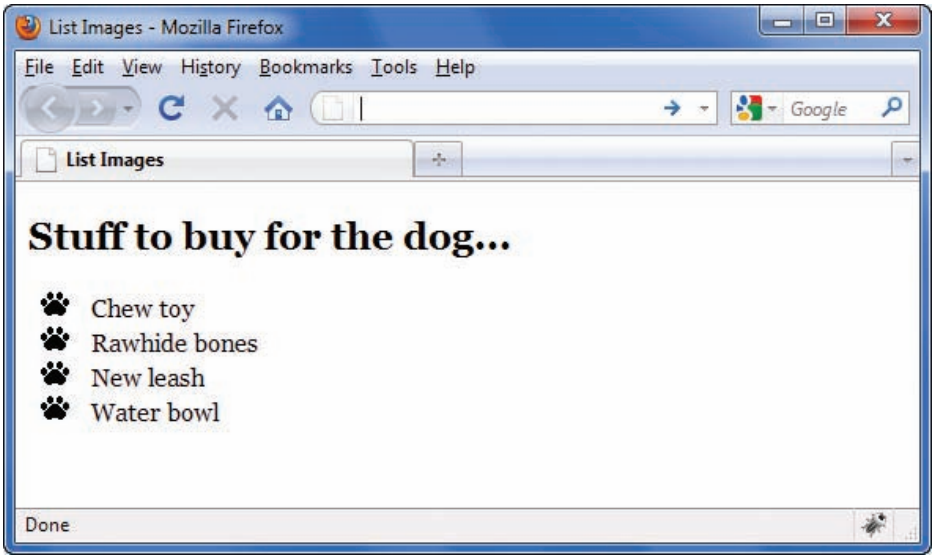


Figure 5-24 Attaching an image as a list marker

Specifying the list-style-position Property

The list-style-position property lets you determine the placement of the list marker, either inside or outside the list-item content box.

list-style-position	
Value:	inside | outside | inherit
Initial:	outside
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual

The default value is outside. Figure 5-25 shows the two types of list position values.

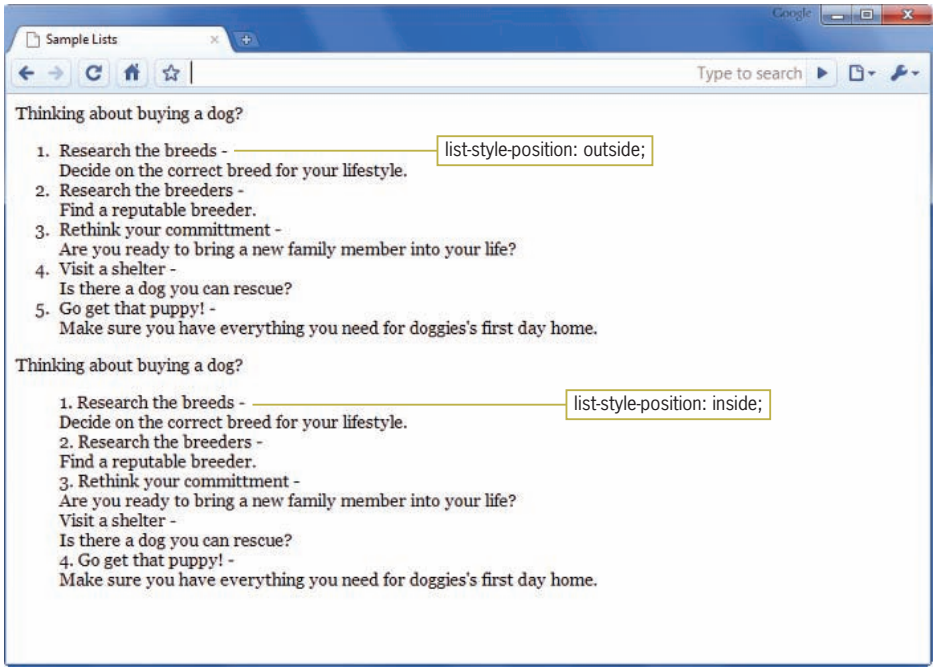


Figure 5-25 Positioning the list marker

This figure shows two <ol> lists. The style rules for this page use a class selector to differentiate between the two:

```
ol.outside {list-style-position: outside;}
ol.inside {list-style-position: inside;}
```

The class is then applied to each <ol> element, as shown in the following code fragment for the first list:

```
<ol class="outside">
```

The class is applied to the second list in the same way:

```
<ol class="inside">
```

Using the list-style Shorthand Property

Like the font shorthand property, the list-style property lets you write a single rule to specify all of the list-item properties.

list-style shorthand property description	
Value:	[<list-style-type> || <list-style-position> || <list-style-image>] | inherit
Initial:	not defined for shorthand properties
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual

The list-style shortcut property lets you state the following list-style properties in one concise style rule:

- List-style-type
- List-style-image
- List-style-position

You can specify values in any order. In the following style rule, the list is set to lowercase alphabetical letters that are inside the list box:

```
ol {list-style: lower-alpha inside;}
```

Chapter Summary

You can use Cascading Style Sheets to manipulate a variety of text properties and achieve professional-quality typography on your Web site. Keep the following points in mind:

- Use type to communicate information structure. Be sparing with your type choices; use fonts consistently and design for legibility.
- Remember that HTML text downloads faster than graphics-based text. Use HTML text whenever possible.
- Use fonts that appear as consistently as possible across operating systems.
- Standardize your styles by building external style sheets and linking them to multiple documents.
- Test your work. Different browsers and computing platforms render text in different sizes.
- Use type effectively by choosing available fonts and sizes. Design for legibility and use text to communicate information about the structure of your material.

- Choose the correct measurement unit based on the destination medium. For the computer screen, ems or percentage measurements can scale the text to the user's preferences.
- Use font properties to control the look of your letter forms. Specify font-substitution values to ensure that your text is displayed properly across different platforms.
- Use the text spacing properties to create more visually interesting and legible text.

Key Terms

cursive—A generic value for the CSS font-family property. Cursive fonts are designed to resemble handwriting. Most browsers do not support this font family.

em unit—In CSS, a unit equal to the font size of an element.

ex unit—In CSS, a unit equal to the height of the lowercase letter *x* in any given font.

fantasy—A generic value for the CSS font-family property. Fantasy fonts are primarily decorative. Most browsers do not support this font family.

font—A typeface in a particular size, such as Times Roman 24 point.

font property—In CSS, a shortcut that lets you specify the most common font properties in a single statement.

leading—The white space between lines of text.

monospace—A generic value for the CSS font-family property. Monospace fonts are fixed-width fonts. Every letter has the same horizontal width.

sans-serif—A generic value for the CSS font-family property. Sans-serif fonts have no serifs. The most common sans-serif fonts are Helvetica and Arial.

serif—A generic value for the CSS font-family property. Serif is the traditional printing letter form, with strokes (or serifs) that finish off the top and bottom of the letter. The most common serif fonts on the Web are Times and Times Roman.

text property—A CSS property that lets you adjust the spacing around and within your text.

typeface—The name of a type family, such as Times Roman or Futura Condensed.

x-height—The height of the letter *x* in a particular font.

Review Questions And Exercises

1. What is the default browser font?
2. What does the browser do if you specify a font that is not stored on a user's computer?
3. What are two drawbacks to the use of graphics-based text?
4. What are the three types of CSS measurement units?
5. What is the best destination for absolute units of measurement?
6. Why would you use relative or percentage values for a Web page?
7. What is the size of the em?
8. What determines the size of a pixel?
9. What is the advantage of the generic font families?
10. Write a font-family substitution string that selects Arial, Helvetica, or any sans-serif font for a <p> element.
11. Write a style rule for an <h2> element that specifies bold text that is twice the size of the default font size.
12. Write a rule specifying that <p> elements appear as 1.5em text with 2em leading.
13. Write a rule specifying that elements are displayed in red only when they appear within <p> elements.
14. Write a rule defining a division named *note*. Specify 12-point bold Arial text on a yellow background.

15. What three typographic white-space areas can you affect with style rules?

16. Write a style rule for a <p> element with a 25-point hanging indent and a 30-pixel margin on the left and right sides.

17. Rewrite the following rule using the font shortcut property:

```
blockquote {font-style: italic; font-size: 1.2em;
line-height: 1.75em; font-family: times, serif;}
```

18. What is a benefit of increasing the standard text line height?

19. What is the size of the text indent and line height relative to the user's default font size in the following style rule?

```
p {text-indent: 3em; line-height: 150%;}
```

Hands-On Projects

1. In the following set of steps, you will learn how to style list-item elements with the list-style properties. As you work through the exercise, refer to Figure 5-27 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   To apply the list-style properties:

   a. Open the file **lists.html** in your HTML editor, and save it in your work folder as **lists1.html**.

   b. Copy the image file **diamond.gif** into your work folder.

   c. In your browser, open the file **lists1.html**. When you open the file, it looks like Figure 5-26. Notice that the file contains three lists. You will apply a different list-style to each list.

   d. The first list on the page is a bulleted list that currently displays the default disc (bullet) style. Write a style rule that uses a class selector *circle* to uniquely select the list. Set the list-style-type property to change the bullet style to *circle*.

      ```
      ul.circle {list-style-type: circle;}
      ```

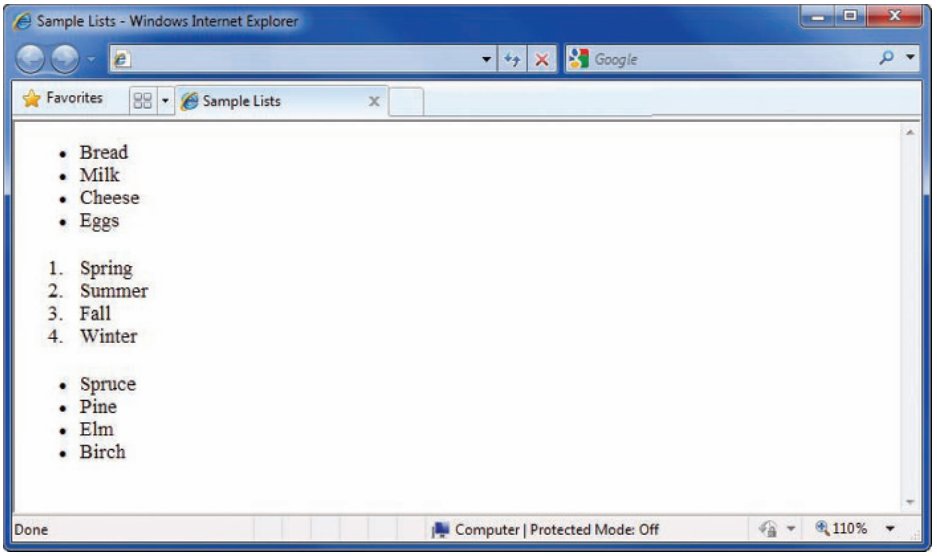


Figure 5-26 Beginning Web page

e. Now apply the style to the first list by adding the *class* attribute to the <ul> element.

```
<!-- Bulleted List -->
<ul class="circle">
<li>Bread</li>
<li>Milk</li>
<li>Cheese</li>
<li>Eggs</li>
</ul>
```

f. The second list on the page is an ordered list that currently displays the default decimal style. Write a style rule that uses a class selector *alpha* to uniquely select the list. Set the list-style-type property to change the style to upper-alpha.

```
ol.alpha {list-style-type: upper-alpha;}
```

g. Now apply the style to the first list by adding the *class* attribute to the <ol> element.

```
<!-- Alphabetical List -->
<ol class="alpha">
<li>Spring</li>
<li>Summer</li>
<li>Fall</li>
<li>Winter</li>
</ol>
```

h. The third list on the page is an unordered list that currently displays the default bullet style. Write a style rule that uses a class selector *image* to uniquely select the list. Set the list-style-image property to a URL value, using the image file diamond.gif.

```
ul.image {list-style-image: url(diamond.gif);}
```

i. Now apply the style to the first list by adding the *class* attribute to the <ol> element. Figure 5-27 shows the finished document.

```
<!-- List Image -->
<ul class="image">
<li>Spruce</li>
<li>Pine</li>
<li>Elm</li>
<li>Birch</li>
</ul>
```

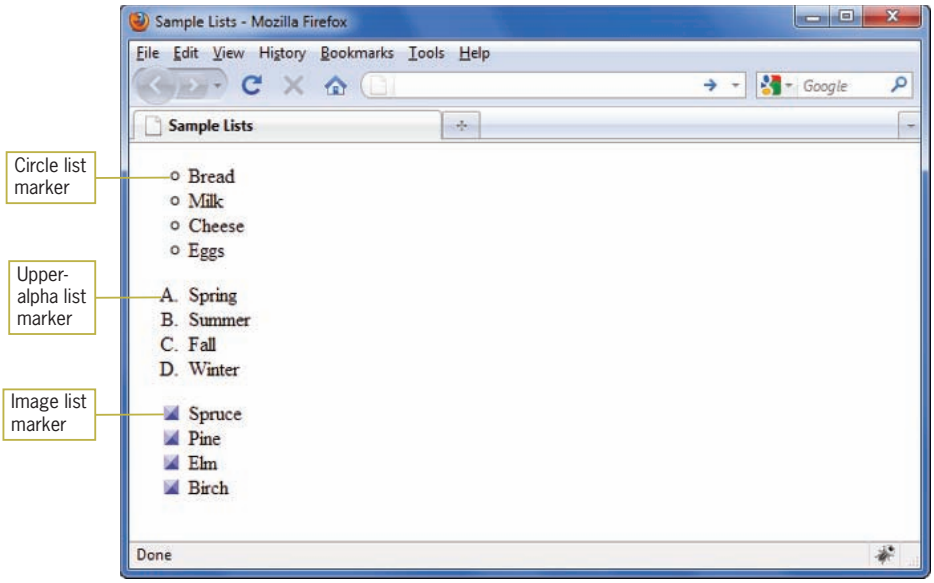


Figure 5-27 Completed Web page in Firefox

2. Modify an existing HTML document to use Cascading Style Sheets.

   a. Build styles using the existing standard HTML elements in the file.

   b. Test the work in multiple browsers to verify that all styles are portable.

c. Remove the files and place them in an external style sheet.

d. Link the HTML file to the style sheet. Test to make sure the file is displayed properly.

3. Browse the Web for examples of good typography. Write a short design critique of why the type works effectively on the Web sites you find. Save and print screen shots of the sample Web pages to accompany your critique.

4. Browse the Web for examples of poor typography. Write a short design critique of why the type is confusing or misleading to the user. Save and print screen shots of the sample Web pages to accompany your critique.

5. In this project, you have a chance to test the font and text properties on paragraphs of text. Save and view the file in your browser after completing each step.

   a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains multiple <p> elements and so on. Save the file in your Chapter05 folder as **fonts1.html**.

   b. Add a <style> element to the <head> section, as shown in the following code:

```
<head>
<title>CSS Test Document</title>
<style type="text/css">

</style>
</head>
```

   c. Write a style rule that uses p as a selector and sets the font-family to a sans-serif font. You can use a generic font family, or choose one of the fonts available on your computer.

   d. Specify a list of alternate fonts to ensure that your font choice is displayed properly across a range of computers.

   e. Specify a text indent for the <p> elements. Use the em value as the measurement unit.

   f. Add the line-height property to the style rule. Experiment with different line heights until you find one that you feel enhances the legibility of the paragraph text.

Individual Case Project

Design the type hierarchy for your Case Project Web site. Create a type specification HTML page that shows examples of the different typefaces and sizes and how they will be used. This can be a mock-up page that uses generic content but demonstrates the overall typographic scheme. Consider the following questions:

- What will be the typefaces and styles for the body type and headings?
- How many levels of headings are necessary?
- What are the different weights and sizes of the headings?
- How will text be emphasized?
- Will hypertext links be standard or custom colors?
- How will you ensure the legibility and readability of your text?
- What will your line length be?

Team Case Project

Meet as a team and discuss the type hierarchy and options for your site. Each team member should bring a type specification HTML page with his or her ideas and examples of the different typefaces and sizes and how they can be used. This can be a mock-up page that uses generic content but demonstrates the overall typographic scheme. Consider the following questions:

- What will be the typefaces and styles for the body type and headings?
- How many levels of headings are necessary?
- What are the different weights and sizes of the headings?
- How will text be emphasized?
- Will hypertext links be standard or custom colors?
- How will you ensure the legibility and readability of your text?
- What will your line length be?

CHAPTER 6

Box Properties

When you complete this chapter, you will be able to:

- Understand the CSS visual formatting model
- Use the CSS box model
- Apply the margin properties
- Apply the padding properties
- Apply the border properties
- Use the page layout box properties
- Create a simple page layout

In this chapter, you will explore the CSS box properties. These properties let you control the margin, padding, and border characteristics of block-level elements. To understand how these properties work, you will first learn about the CSS visual formatting model and the box model. These models control the way content is displayed on a Web page. Then you will learn about the margin, padding, and border properties and how you can use them to enhance the display of content in the browser. Finally, you will see how the special box properties—width, height, float, and clear—let you create containers for content that you can position on your Web page. These box properties will become the basis for the page layout techniques you will learn about in the next chapter.

Understanding the CSS Visual Formatting Model

The **CSS visual formatting model** describes how the element content boxes should be displayed by the browser, for example, whether scroll bars appear and how text is wrapped based on the browser window size. The visual formatting model is based on the hierarchical structure of the HTML document and the element display type. In HTML, elements fall into two primary box types:

- *Block*—Block-level boxes appear as blocks such as paragraphs. Block elements can contain other block elements or inline boxes that contain the element content.
- *Inline*—Inline-level boxes contain the content within the block-level elements. They do not form new blocks of content.

Figure 6-1 shows three different block-level elements: <body>, <h1>, and <p>. The <h1> and <p> elements contain inline content boxes.

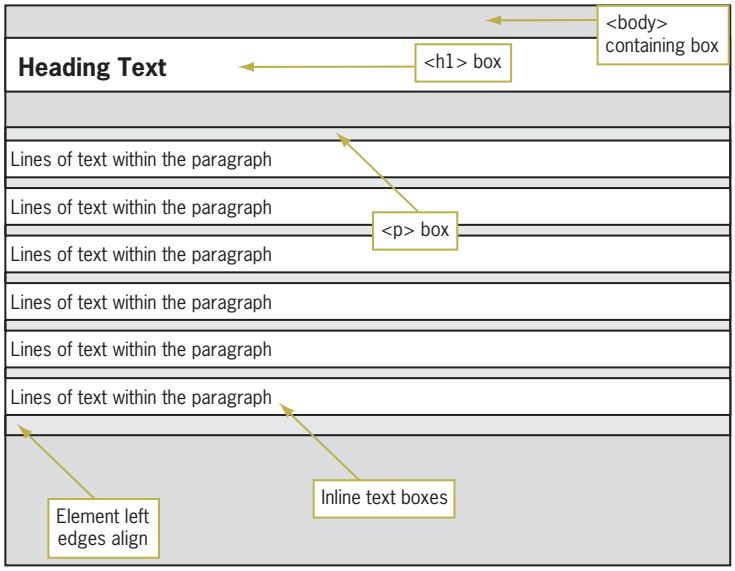


Figure 6-1 The CSS visual formatting model

Figure 6-1 also shows that parent elements contain child elements. The parent element is called the **containing box**. You can see that <body> is the containing box for the elements of a Web page. All other elements' boxes reside within <body>, some of which may contain their own child boxes. In Figure 6-1, the <body> element is the containing box for the <h1> and <p> elements. The <p> element is the containing box for the inline text that comprises the paragraph text. Inline text boxes are split as necessary to fit the dimensions of the containing box and to wrap text to the next line.

CSS lets you specify margin, border, and padding values for all block-level elements. In some instances, the values you specify depend on the containing box that is the parent of the element you want to affect. For example, if you choose a percentage value for a margin, the percentage value is based on the containing box. In Figure 6-1, a 10% margin value for the <p> element would create margins that are 10% of the width of the containing box, in this case, the <body> element.

CSS lets you change the display type of any box, either block or inline, with the display property, described next.

Specifying the Display Type

display property description	
Value:	block | inline | list-item | none | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none
Initial:	inline
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The CSS display property determines how the browser displays an element. The display property values let you create block-level, inline, and list type elements. It also specifies values for a CSS table model, such as *table* and *table-row*, which are currently not supported by all browsers. The two most commonly used values are *block* and *inline*.

The display property is often used to create horizontal navigation lists, which you will learn about in Chapter 9. Here is a simple example that uses the display property with an <li> selector to make the list items appear horizontally across the page:

```
li {
   display: inline;
   list-style-type: none;
}
```

The list-style-type property hides the bullets that are normally displayed with an unordered list, as you saw in Chapter 5. The style rule applies to the following HTML code for an unordered list:

```
<ul>
   <li><a href="url">Home</a></li>
   <li><a href="url">Search</a></li>
   <li><a href="url">Products</a></li>
   <li><a href="url">Contact Us</a></li>
   <li><a href="url">Support</a></li>
   <li><a href="url">Downloads</a></li>
</ul>
```

The result of this style rule are list elements that can be used to create a navigation bar as shown in Figure 6-2.

The display property *none* value lets you hide an element so that it is not displayed in the browser. This can be useful in a Web page where you want to display only some of the information on a page, or with JavaScript to create elements that can be displayed as a result of user action.

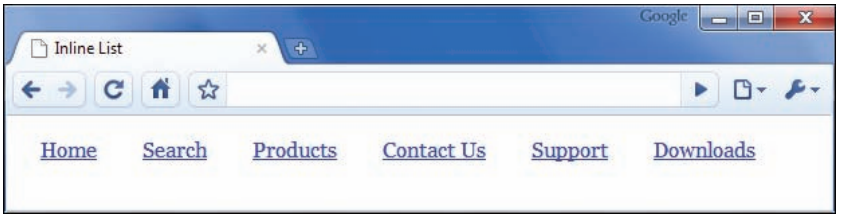


Figure 6-2 Horizontal list created with the display property

Using the CSS Box Model

The CSS **box model** describes the rectangular boxes that contain content on a Web page. Each block-level element you create is displayed in the browser window as a box with content. Each content box can have margins, borders, and padding, as shown in Figure 6-3.

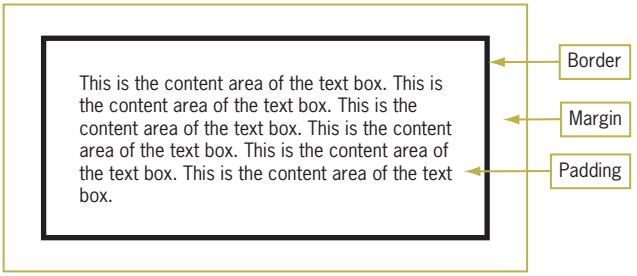


Figure 6-3 CSS box model

As Figure 6-3 illustrates, the content box is the innermost box, surrounded by the padding, border, and margin areas. The padding area has the same background color as the content element, but the margin area is always transparent. The border separates the padding and margin areas.

Figure 6-4 shows the box model areas in a paragraph element. This paragraph has 2em padding, a thin black border, and 2em margins. Notice that the margin area is transparent.

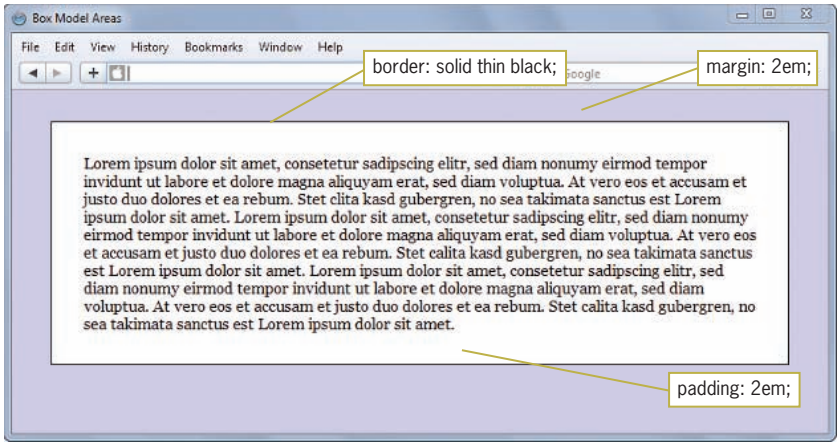


Figure 6-4 CSS box model areas in a <p> element

The following code shows the style rule for the paragraph in Figure 6-4:

```
p {
   margin: 2em;
   padding: 2em;
   border: solid thin black;
   background-color: white;
}
```

The margin and padding properties set the length to 2em for all four sides of the box. The border property sets the border-style to solid, the border-weight to thin, and the border-color to black. The background-color property sets the paragraph background color to white.

CSS lets you specify margin, padding, and border properties individually for each side of the box. Figure 6-5 shows that each area has a left, right, top, and bottom side. Each one of the sides can be referred to individually. If the browser supports the individual properties, you can select, for example, the padding-bottom, border-top, or margin-left properties if you prefer.

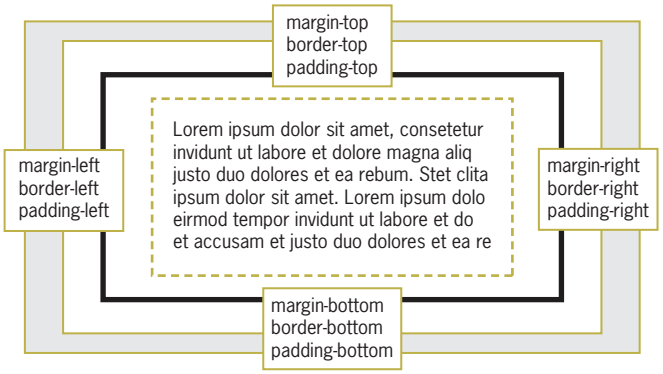


Figure 6-5 CSS box model individual sides

Figure 6-6 shows a paragraph with a variety of box property settings. As you can see, CSS gives you complete control over the individual white space areas and borders in block elements. The style rule for the paragraph is:

```
p {
   background-color: white;
   border-left: 6px solid;
   margin-left: 2em;
   margin-top: 3em;
   padding-top: 2em;
   padding-right: 2em;
   padding-bottom: 1em;
   padding-left: 1em;
}
```

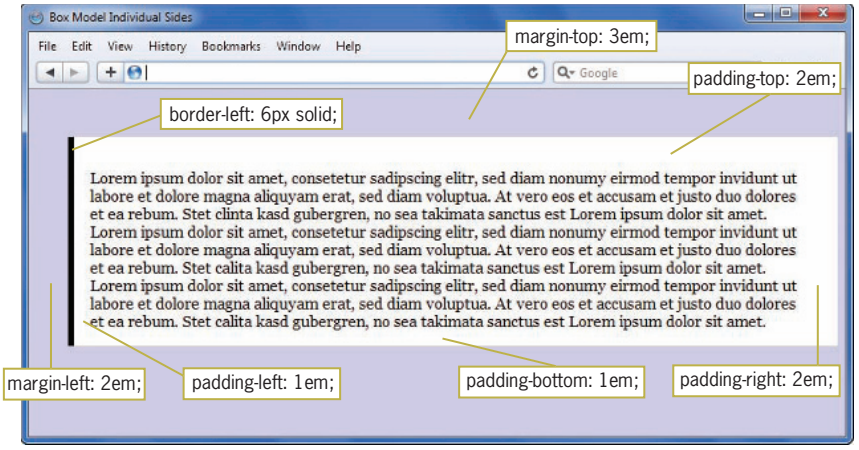


Figure 6-6 CSS box model individual sides in a <p> element

Measurement Values

The margin, border, and padding properties let you state two types of measurement values—either a length or a percentage. (For a full discussion of measurement values, see Chapter 5.) If you use a percentage value, the percentage is based on the width of the containing box, as described earlier. If you choose a length, you have to decide whether to use an absolute or relative value. As with font sizes, you are better off using relative units such as ems or percentages when you are stating margin, border, or padding sizes. The relative measurement values let you build scalable Web pages. Many Web designers prefer pixel measurements for certain properties such as borders when they know the size of the border line will remain constant across different monitors and devices. Pixel values for padding and margins are often specified in fixed page designs where the dimensions of the layout are constant.

Always use relative measurement values, such as ems or percentages, if you want your Web pages to adapt to different browser sizes or user-applied font sizes.

Applying the Margin Properties

The **margin properties** let you control the margin area of the box model. Margins are always transparent, showing the background of their containing element. You can use margins to enhance the legibility of text, create indented elements, and add white space around images.

Specifying Margins

margin, margin-top, margin-right, margin-bottom, margin-left, property description	
Value:	<length> | <percentage>
Initial:	0
Applies to:	all elements
Inherited:	no
Percentages:	refer to width of containing block

The margin properties let you specify margins with either a length or percentage value. You can use the margin property to state one value for all four margin sides, or you can specify settings for individual margins. The following style rule sets all four margins in a paragraph to 2em.

```
p {margin: 2em;}
```

You can also choose to specify individual margin properties that let you control each margin: margin-left, margin-right,

margin-top, and margin-bottom. The following style rules set the left and right margins for a paragraph element:

```
p {
   margin-left: 2em;
   margin-right: 3em;
}
```

Margin Property Shorthand Notation

The shorthand notation syntax lets you state individual margin settings within the same rule. The individual margin settings change based on the number of values and their order within the rule. Table 6-1 shows how the syntax works.

Number of Values	Example	Description
1 value	p {margin: 1em;}	All four margins are 1em
2 values	p {margin: 1em 2em;}	Top and bottom margins are 1em Left and right margins are 2em
3 values	p {margin: 1em 2em 3em;}	Top margin is 1em Right and left margins are 2em Bottom margin is 3em
4 values	p {margin: 1em 2em 3em 4em;}	Top margin is 1em Right margin is 2em Bottom margin is 3em Left margin is 4em

Table 6-1 Shorthand Notation for the margin Property

It is your choice whether you want to use the shorthand notation or to state the margin properties individually. Some designers prefer the more specific and easy to read individual margin properties rather than the shorthand notation. Others prefer the brief syntax of the shorthand notation. Either way, set a coding convention for your Web site and use it consistently.

Figure 6-7 shows two paragraph elements. The first paragraph has the default margin setting; the second has the margin set to 2em. Notice that the increased margins enhance the legibility of the text.

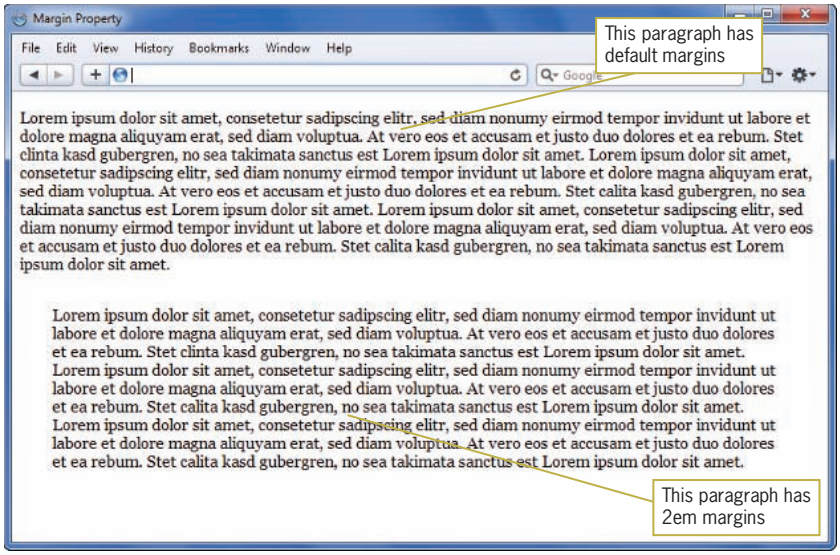


Figure 6-7 Using the margin property

Negative Margins

Margins are unusual because you can use negative values. You can set negative margin values to achieve special effects. For example, you can remove the default margins by setting a negative value or overlap elements on the page.

Figure 6-8 shows two <h1> elements, one with the default margins, and one with the bottom margin set to a negative value. The following rule sets a negative value of 20 pixels for the element's bottom margin:

```
h1 {margin-bottom:-20px;}
```

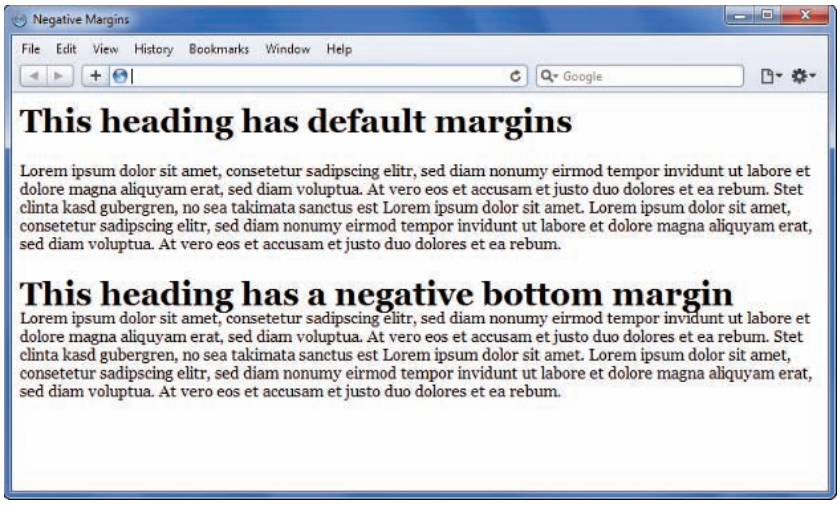


Figure 6-8 <h1> element with a negative bottom margin

Collapsing Margins

To ensure that the spacing between block-level elements is consistent, the browser collapses the vertical margins between elements. The vertical margins are the top and bottom element margins. The browser does not add the value of the two, but picks the greater value and applies it to the space between the adjoining elements. To illustrate this, consider the following rule:

```
p {
   margin-top: 15px;
   margin-bottom: 25px;
}
```

If the browser did not collapse the vertical margins, the paragraphs would have 40 pixels of space between each paragraph. Instead, the browser collapses the margin. Following the CSS convention, the browser sets the vertical margin between paragraphs to 25 pixels, the greater of the two values. Figure 6-9 shows how the browser maintains the space between the paragraphs.

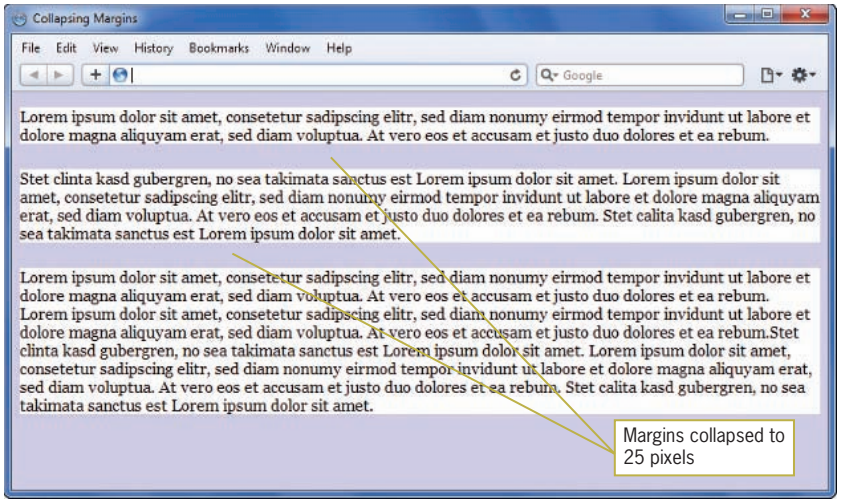


Figure 6-9 Browser collapses vertical margins

Zeroing Margins

You can set margin values to zero if you want to remove the default margin spacing that is built into the browser. This technique is useful if you have problems getting your page designs to look consistent across multiple browsers and display devices. By setting the values in the body section to zero, the settings are

inherited for all elements on the page. Then you can set specific margin settings for each element as necessary for your design. The style rule for zeroing margins looks like this:

```
body {margin: 0; padding: 0;}
```

This technique is also handy when you want to place an image against the side of the browser window as shown in Figure 6-10.

Figure 6-10 Zeroing margins to place images against the edge of the window

Applying the Padding Properties

The CSS **padding properties** let you control the padding area in the box model. The padding area is between the element content and the border. The padding area inherits the background color of the element, so if a <p> element has a white background, the padding area will be white as well.

If you zero margins for the entire page, you must explicitly set margins for any elements that you do not want to place against the edge of the browser window.

padding, padding-top, padding-right, padding-bottom, padding-left property descriptions

Value:	<length> | <percentage>
Initial:	0
Applies to:	all elements
Inherited:	no
Percentages:	refer to width of containing block

Figure 6-11 shows how adding padding to an element increases the space between the content and the edge of the element box.

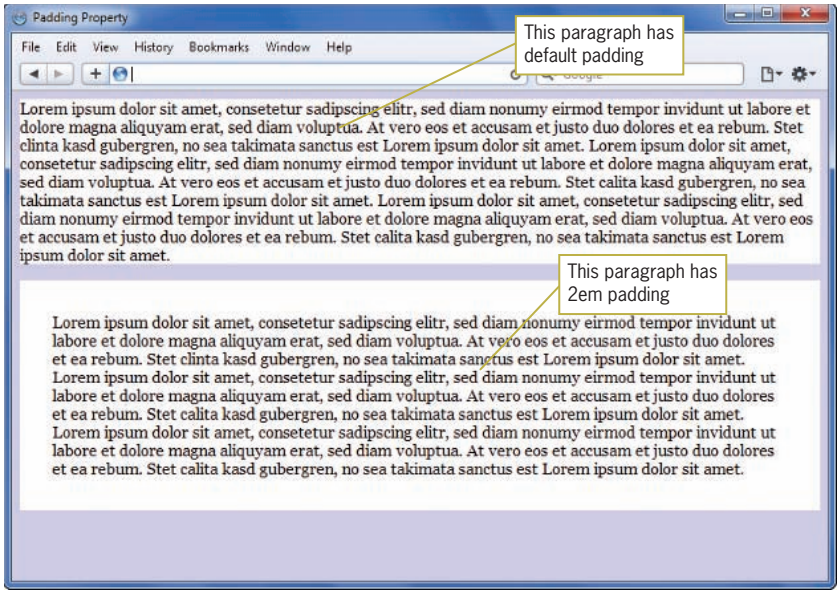


Figure 6-11 Default padding and 2em padding

Like the margin property, the padding property lets you state one value for all four margin sides, or you can specify settings for individual padding areas. You can specify either a length or a percentage value. Unlike margins, you cannot collapse the padding area or set negative padding values. The following style rule sets all four padding sides in a paragraph to 2em.

```
p {padding: 2em;}
```

You can also choose to specify individual padding properties, which let you control each padding area: padding-left,

padding-right, padding-top, and padding-bottom. The following style rules set the left and right padding for a paragraph element:

```
p {
   padding-left: 2em;
   padding-right: 3em;
}
```


Padding Property Shorthand Notation

The shorthand notation syntax lets you state individual padding settings. Like the margin shorthand property described earlier, the individual padding settings change based on the number of values and their order within the rule. Table 6-2 shows how the syntax works.

Number of Values	Example	Description
1 value	p {padding: 1em;}	Top, bottom, left, and right padding are 1em
2 values	p {padding: 1em 2em;}	Top and bottom padding are 1em Left and right padding are 2em
3 values	p {padding: 1em 2em 3em;}	Top padding is 1em Right and left padding are 2em Bottom padding is 3em
4 values	p {padding: 1em 2em 3em 4em;}	Top padding is 1em Right padding is 2em Bottom padding is 3em Left padding is 4em

Table 6-2 Shorthand Notation for the padding Property

The following style rule sets the top and bottom padding areas for a paragraph, along with complementing borders, a white background, and margins to offset the paragraph from the browser sides:

```
p {
   padding-top: 2em;
   padding-bottom: 2em;
   border-top: solid thin black;
   border-bottom: solid thin black;
   background-color: white;
   margin: 2em;
}
```

As Figure 6-12 shows, the paragraph now has the default left and right padding with 2em top and bottom padding.

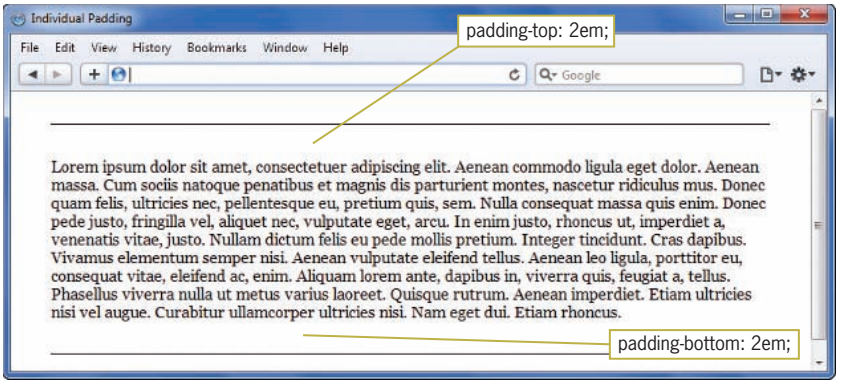


Figure 6-12 Using the individual padding properties

Applying the Border Properties

The **border properties** let you control the appearance of borders around elements. The border area resides between the margin and padding. Border properties for each border side let you specify the style, width, and color of each border. You will most likely use the five border shorthand properties, which include:

- border
- border-left
- border-right
- border-top
- border-bottom

These shorthand properties let you state border-style, border-color, and border-width for all four borders or for any of the individual sides of the box. However, you can also state much more specific borders by using the border properties separately. Table 6-3 lists the entire range of 20 border properties.

Description	Property Name		
Overall shorthand property	border		
Individual side shorthand properties	border-left, border-top, border-right, border-bottom		
Specific shorthand property	border-style	border-width	border-color
Individual properties	border-left-style	border-left-width	border-left-color
	border-right-style	border-right-width	border-right-color
	border-top-style	border-top-width	border-top-color
	border-bottom-style	border-bottom-width	border-bottom-color

Table 6-3 Border Properties

Before you can use the shorthand properties, you must first understand the three border characteristics—border-style, border-color, and border-width.

Specifying Border Style

border-style, border-top-style, border-right-style, border-bottom-style, border-left-style property description

Value:	<border-style>
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The border-style is the most important border property because it must be stated to make a border appear. The border-style property lets you choose from one of the following border style keywords:

- *none*—No border on the element; this is the default setting
- *dotted*—Dotted border
- *dashed*—Dashed border
- *solid*—Solid line border
- *double*—Double line border
- *dot-dash*—Alternating dots and dashes (CSS3 value)
- *dot-dot-dash*—Two dots and a dash (CSS3 value)
- *wavy*—Wavy line border (CSS3 value)
- *groove*—Three-dimensional border that appears to be engraved into the page
- *ridge*—Three-dimensional border that appears to be embossed (or extend outward from the page)
- *inset*—Three-dimensional border that appears to set the entire box into the page
- *outset*—Three-dimensional border that appears to extend the entire box outward from the page

The following code shows an example of the border-style property in use:

```
p {border-style: solid;}
```

Figure 6-13 shows examples of the borders. As you can see, each browser displays the border styles differently. If a border you specify is not supported, the border defaults to solid.

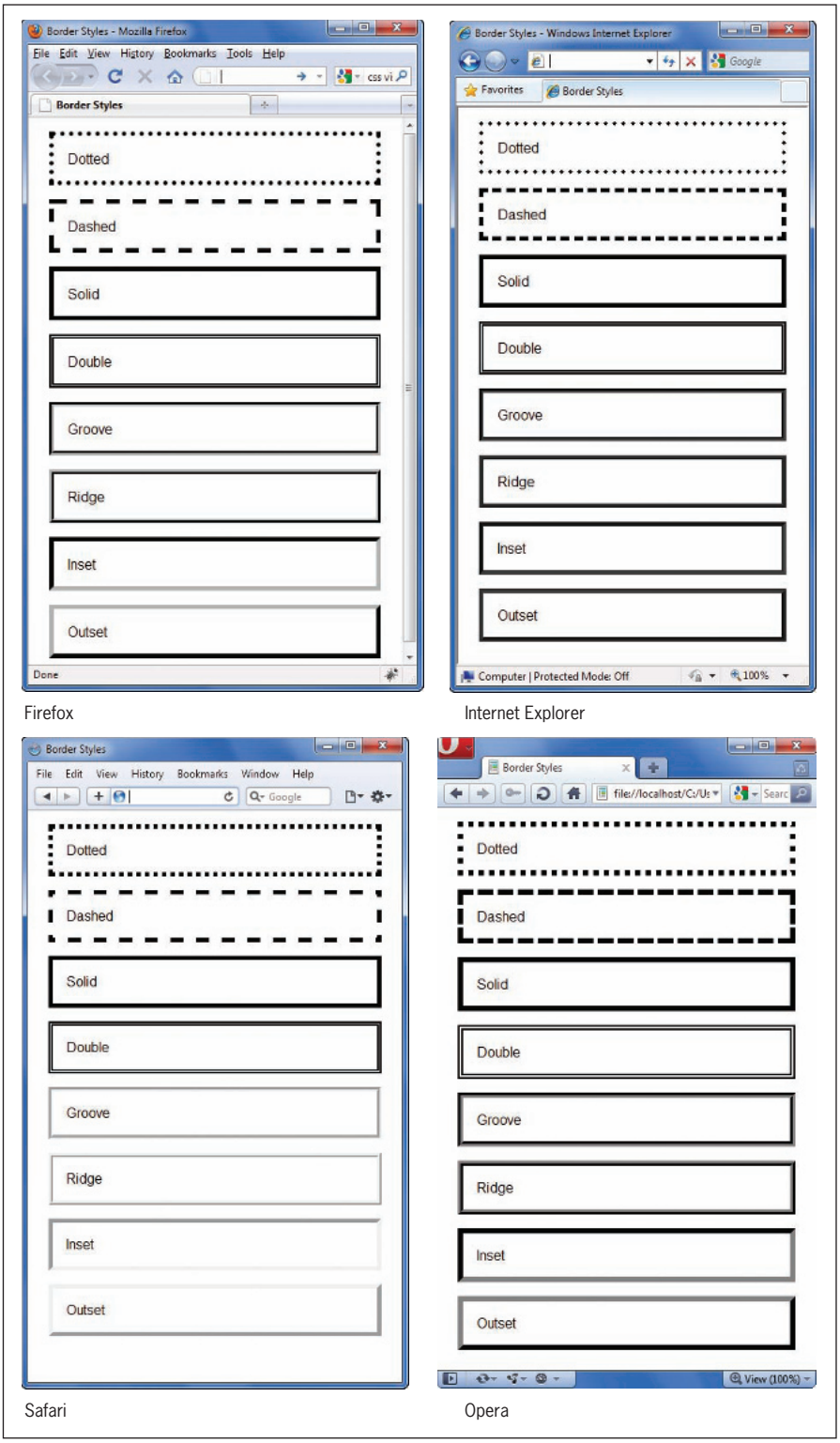


Figure 6-13 Border styles in Firefox, Internet Explorer, Safari, and Opera

Individual Border Styles

You can also specify individual borders styles with the following border-style properties:

- border-left-style
- border-right-style
- border-top-style
- border-bottom-style

These properties let you single out one border and apply a style. The following rule applies only to the left border of the element:

```
p {border-left-style: double;}
```

Border-Style Property Shorthand Notation

The shorthand notation syntax lets you state individual border-style settings. The individual border style settings change based on the number of values and their order within the rule. Table 6-4 shows how the syntax works.

Number of Values	Example	Description
1 value	p {border-style: solid;}	All four borders are solid
2 values	p {border-style: solid double;}	Top and bottom borders are solid Left and right borders are double
3 values	p {border-style: solid double dashed;}	Top border is solid Right and left borders are double Bottom border is dashed
4 values	p {border-style: solid double dashed dotted;}	Top border is solid Right border is double Bottom border is dashed Left border is dotted

Table 6-4 Shorthand Notation for the border-style Property

Of course, if you examine the rules in Table 6-4, you can see they will create odd effects. For example, a paragraph with a different border style for each side is not a common design technique. Remember to use restraint and keep the user in mind when working with border styles.

Specifying Border Width

border-width, border-top-width, border-right-width, border-bottom-width, border-left-width property description	
Value:	thin | medium | thick | <length>
Initial:	medium
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The border-width property lets you state the width of the border with either a keyword or a length value. You can use the following keywords to express width:

- thin
- medium (default)
- thick

When designing layouts, it is often handy to turn the borders on for an element, even though you may turn it off for the finished Web page. The border lets you see the dimensions of the element on the page. Remember that adding borders to an element increases the element's width.

The width of the rule when you use these keywords is based on the browser. The length values let you state an absolute or relative value for the border; percentages are not allowed. Using a length value lets you create anything from a hairline to a very thick border. The following code shows an example of the border-width property in use:

```
p {border-width: 1px; border-style: solid;}
```

Remember that the border is not displayed unless the border-style property is stated. Figure 6-14 shows examples of different border widths.

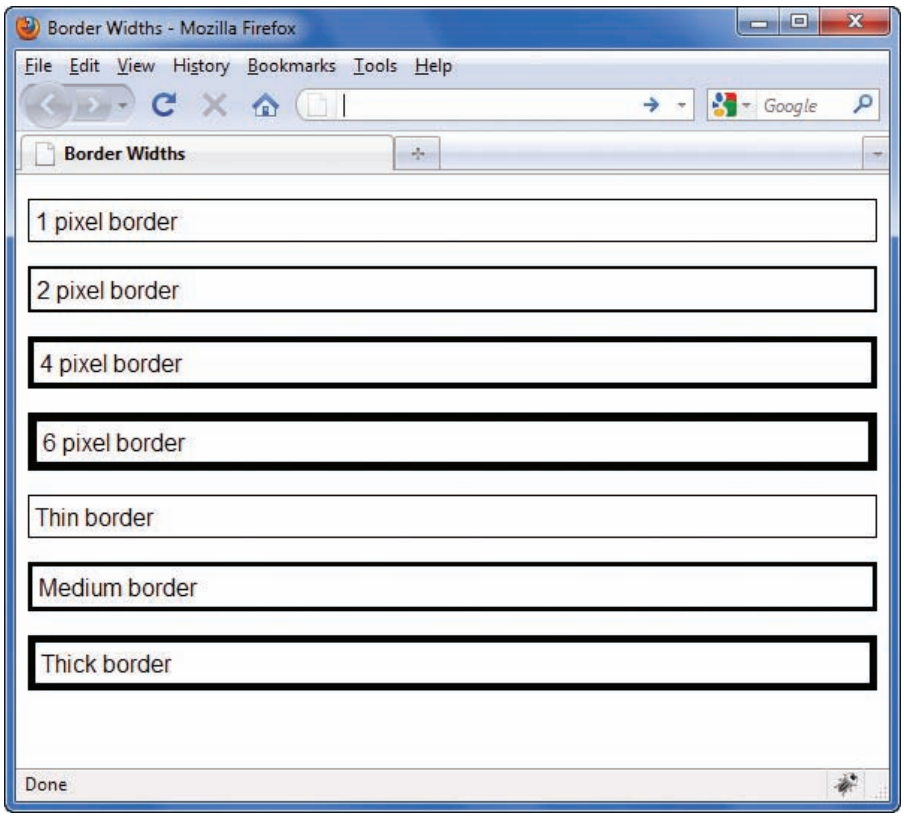


Figure 6-14 Different border widths

Individual Border Widths

You can also specify individual borders widths with the following border-width properties:

- border-left-width
- border-right-width
- border-top-width
- border-bottom-width

These properties let you single out one border and apply a width. The following rule applies only to the left border of the element:

```
p {border-left-width: thin;}
```

Border-Width Property Shorthand Notation

The shorthand notation syntax lets you state individual border-width settings. The individual border width settings change based on the number of values and their order within the rule. Table 6-5 shows how the syntax works.

Number of Values	Example	Description
1 value	p {border-width: 1px;}	All four borders are 1 pixel wide
2 values	p {border-width: 1px 2px;}	Top and bottom borders are 1 pixel wide
		Left and right borders are 2 pixels wide
3 values	p {border-width: 1px 2px 3px;}	Top border is 1 pixel wide
		Right and left borders are 2 pixels wide
		Bottom border is 3 pixels wide
4 values	p {border-width: 1px 2px 3px 4px;}	Top border is 1 pixel wide
		Right border is 2 pixels wide
		Bottom border is 3 pixels wide
		Left border is 4 pixels wide

Table 6-5 Shorthand Notation for the border-width Property

Specifying Border Color

border-color, border-top-color, border-right-color, border-bottom-color, border-left-color property description

Value: <color>
Applies to: all elements
Inherited: no
Percentages: N/A

The border-color property lets you set the color of the element border. The value can be either a hexadecimal value, RGB value, or one of the 16 predefined color names shown in the following list:

Aqua	Navy	Gray	Silver
Black	Olive	Green	Tea
Blue	Purple	Lime	White
Fuchsia	Red	Maroon	Yellow

You can learn more about color values in Chapter 8.

To set a border color, use the property as shown in the following rule:

```
p {border-color: red; border-width: 1px; border-style: solid;}
```

The default border color is the color of the element content. For example, the following style rule sets the element text color to red. The border is also red because a border color is not specified.

```
p {
   color: red;
   font: 12pt arial;
   border: solid;
}
```

Individual Border Colors

You can also specify individual border colors with the following border-color properties:

- border-left-color
- border-right-color
- border-top-color
- border-bottom-color

These properties let you single out one border and apply a color. The following property applies only to the left border of the element:

```
p {border-left-color: red; border-style: solid;}
```

Border-Color Property Shorthand Notation

The shorthand notation syntax lets you state individual border-color settings. The individual border color settings change based on the number of values and their order within the rule. You can also choose to state individual border colors in the border-color property. The individual border color settings change based on the order within the rule. Table 6-6 shows how the syntax works.

Number of Values	Example	Description
1 value	p {border-color: black;}	All four borders are black
2 values	p {border-color: black red;}	Top and bottom borders are black Left and right borders are red
3 values	p {border-color: black red green;}	Top border is black Right and left borders are red Bottom border is green
4 values	p {border-color: black red green blue;}	Top border is black Right border is red Bottom border is green Left border is blue

Table 6-6 Shorthand Notation for the border-color Property

Using the Border Shorthand Properties

The shorthand properties are the most common and easiest way to express border characteristics. When you use these shorthand properties, you are stating the style, color, and width of the border in one concise rule.

Border, border-top, border-right, border-bottom, border-left property description

Value:	<border-width> | <border-style> | <color>
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The border property lets you state the properties for all four borders of the element. You can state the border-width, border-style, and border-color in any order. Border-style must be included for the border to appear. If you do not include border-width, the width defaults to medium. If you do not include border-color, the border appears in the same color as the element. The following example rules show different uses of the border property.

The following rule sets the border-style to solid. The border-weight defaults to medium. The border-color is the same as the color of the <p> element; because no color is stated, the border color is black.

```
p {border: solid;}
```

The following rule sets the border-style to solid. The border-weight is 1 pixel. The border-color is red.

```
p {border: solid 1px red;}
```

The following rule sets the border-style to double. The border-weight is thin. The border-color is blue. Notice that the order of the values does not matter.

```
p {border: double blue thin;}
```

These let you state border-style, border-width, and border-color in one statement that selects a single element border. For example, the following rule sets border-style to solid and border-weight to thin for both the left and right borders. Because no color is stated, the borders default to the element color.

```
p {border-left: solid thin; border-right: solid thin;}
```

The following rule sets border-style to double and border-color to red for the top border. Because no border-weight is stated, the weight defaults to medium.

```
p {border-top: double red;}
```

Specifying Rounded Borders

border-radius, border-top-right-radius, border-top-left-radius, border-bottom-right-radius, border-bottom-left-radius

Value:	[<length> | <percentage>]
Initial:	0
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The CSS 3 border-radius property lets you create rounded borders on block-level elements, an effect that many Web designers have wanted but were unable to easily create in a standardized way. Although not yet evenly supported by current browsers, the example in Figure 6-15 shows the Chrome browser's version of two styles of rounded corners on a border.

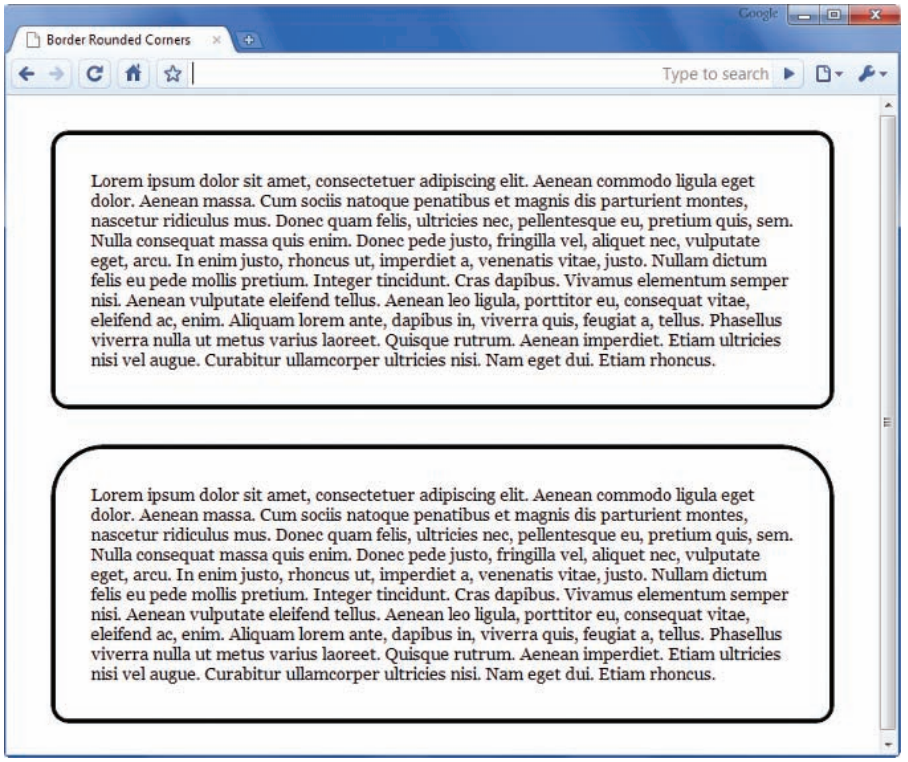


Figure 6-15 Rounded borders

The two length or percentage values determine the radius of each corner. These can be varied to create effects as shown in Figure 6-15.

The following rule sets the radius for all four corners to 1em:

```
border-radius: 1em;
```

The border-radius properties are CSS3 properties that may not work consistently across all browsers, so test your work carefully.

You can also use individual properties to set each corner as shown in the bottom paragraph in Figure 6-15.

```
border-top-right-radius: 3em;
border-top-left-radius: 3em;
border-bottom-right-radius: 1em;
border-bottom-left-radius: 1em;
```

Using the Page Layout Box Properties

The **page layout box properties** let you control the dimensions and position of content boxes. These properties are essential to building CSS page layouts, which you will learn about in Chapter 7. Using these box properties, you can specify the exact shape of a content box and create columns and boxes of content. You can

set minimum and maximum widths and heights as well as control how your boxes are resized based on the size of the browser window. You can also align boxes to the left or right of other elements using the float property and allow text to wrap around images.

In this section, you will learn about the following box properties:

- width, min-width, max-width
- height, min-height, max-height
- float
- clear
- overflow

The element is your best choice for creating content boxes because it can contain other elements such as headings and paragraphs. Class or id names can be applied to the element to apply style rules. As you will see here and in Chapter 7, most Web page layouts are built with elements to create the columns and containers, each of which contains content.

Setting Element Width

width, min-width, max-width property description	
Value:	<length> | <percentage>
Initial:	auto
Applies to:	all elements but text inline elements, table rows, and row groups
Inherited:	no
Percentages:	refer to width of containing block

The width property lets you set the horizontal width of an element using either a length value or a percentage. The percentage value is based on the width of the containing element box. The following is an example of width property usage:

```
div {width: 200px;}
```

If you use percentages, the content boxes will adapt to the size of the browser or the containing element, allowing you to build flexible page layouts based on the browser size. If you are building fixed dimension layouts, use pixel values for width.

Figure 6-16 shows two elements, one that is 200 pixels wide and one that is 50% wide. Each of the elements in this example contains both headings and paragraphs. The left box is fixed at 200 pixels wide. Its height is determined by the content it contains. The right box is set to a percentage value, which means the box size will change based on the size of the browser window. This box's width will always be 50% of the current window size. If the user resizes the browser, the box width will change.

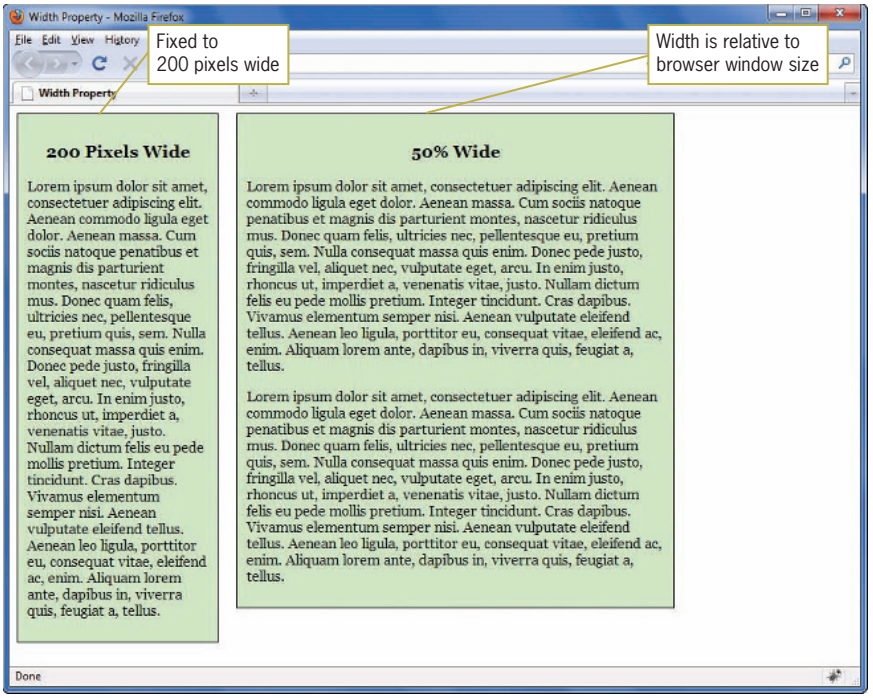


Figure 6-16 Using the width property

Calculating Box Model Width

When you specify a width, you are specifying the width of the content only, not the entire element. In Figure 6-17, the paragraph width is 400 pixels as specified by the width property. The actual element width of 500 pixels is the result of adding the padding, border and margin properties to the content width.

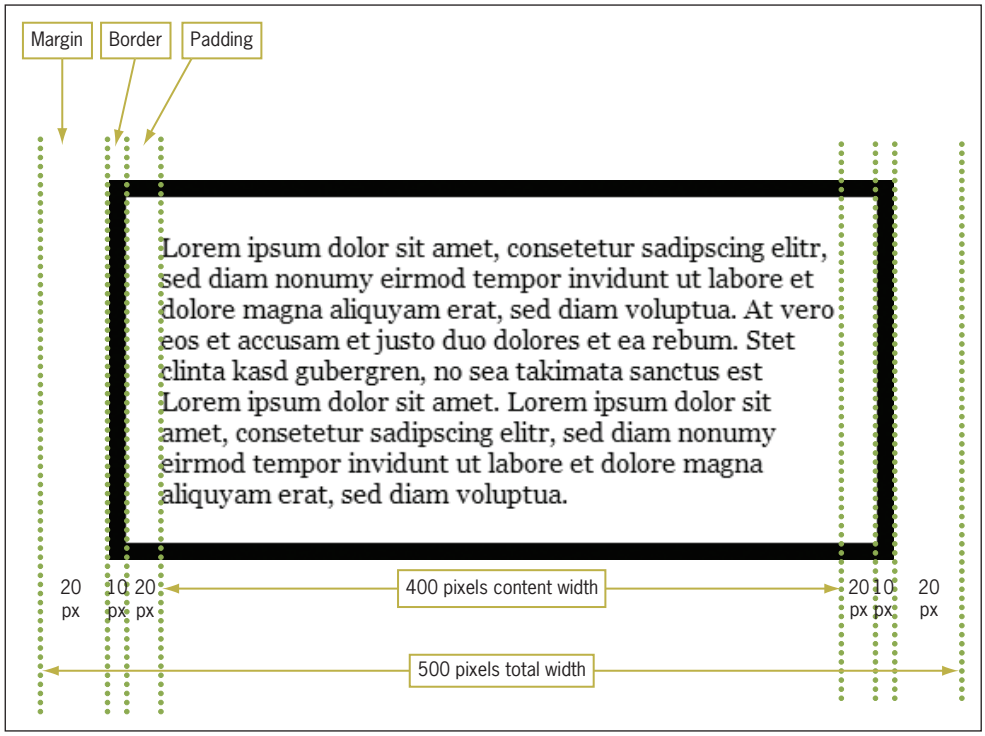


Figure 6-17 Calculating the box model width

Setting Minimum and Maximum Widths

The min-width and max-width properties let you determine exactly how wide or narrow you want the width of a box to be. This is helpful when building flexible layouts, as you will see in Chapter 7. In Figure 6-16, the content element on the right will always be 50% of the browser window, but the user may occasionally make a window very narrow or very wide, and you don't want the box to become either a skinny narrow column or overly wide column. You can control the width with min-width and max-width. In the following example, a page content box is set to 100%, which will fill the browser window. However, the box will not be allowed to shrink to less than 750 pixels or expand to more than 1280 pixels.

```
div.content {
   width: 100%;
   min-width: 750px;
   max-width: 1280px;
}
```

Setting Element Height

height property description	
Value:	<length> | <percentage>
Initial:	auto
Applies to:	all elements but text inline elements, table columns, and column groups
Inherited:	no
Percentages:	N/A

The height property lets you set the vertical height of an element. Height should only be used in situations where you know the exact height of the element content, such as an image. At other times, you may need to create a box with specific dimensions for a design. It is a better practice to let the content determine the height of the element.

The height property accepts either a length value or a percentage. The percentage value is based on the height of the containing element box. The following is an example of height property usage:

```
div {height: 150px; width: 300px;}
```

Setting Minimum and Maximum Height

The min-height and max-height properties let you determine exactly how tall or short you will allow the height of a box to be. These properties work exactly like the min-width and max-width properties described earlier.

Floating Elements

float property description	
Value:	left | right | none
Initial:	none
Applies to:	all elements except positioned elements and generated content (see Appendix B)
Inherited:	no
Percentages:	N/A

The float property lets you position an element to the left or right edge of its parent element. You can float an image to the left or right of text, as shown in Figure 6-18.

Figure 6-18 Floating an image

The style rule for the floating image floats the image to the left and adds a margin to offset the text from the image. The style rule looks like this:

```
img {
   float: left;
   margin-right: 10px;
}
```

The float property can also be used to float a content box to the left or right of text. Used with the width element, you can create a content box as shown in Figure 6-19.

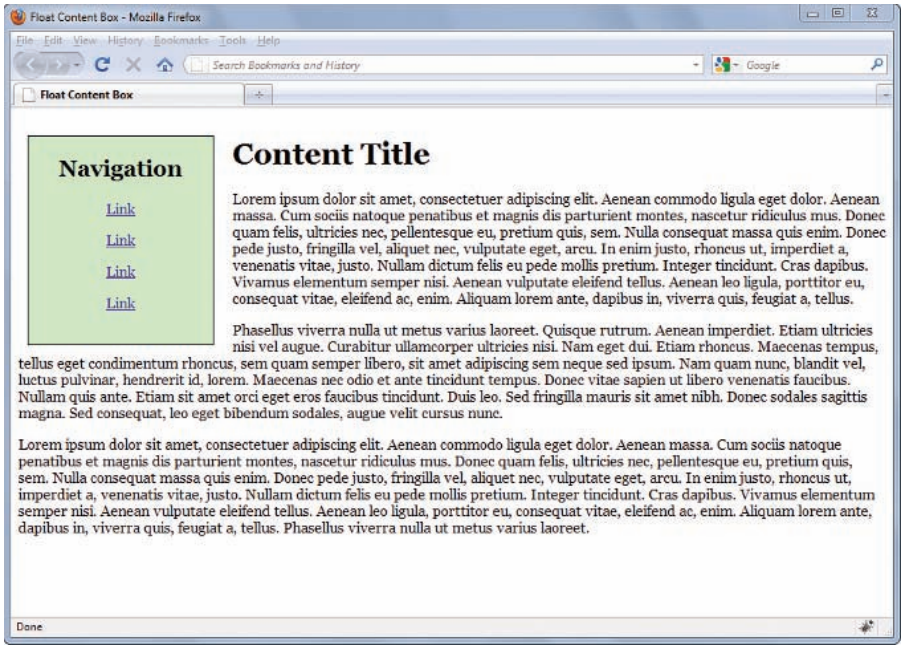


Figure 6-19 Floating content box

The rule for the left-floating content box looks like this:

```
#float {
  width: 200px;
  float: left;
  border: 1px solid black;
  padding-bottom: 20px;
  margin: 0px 20px 10px 10px;
  text-align: center;
  background-color: #cfc;
}
```

The style rule uses an id of *float* as the selector. The width property sets the width of the element to 200 pixels. The float property floats the box to the left of the page. Notice that the rule does not include a height property. The height of the box is determined by the content it contains and the padding values. The margin property states the top, right, bottom, and left margin values.

The floated content box is a element that gets the id value of *float*:

```
<div id="float">
   <h2>Navigation</h2>
   <p><a href="url">Link</a></p>
   <p><a href="url">Link</a></p>
   <p><a href="url">Link</a></p>
   <p><a href="url">Link</a></p>
</div>
```

Clearing Elements

clear property description	
Value:	none | left | right | both
Initial:	none
Applies to:	block-level elements
Inherited:	no
Percentages:	N/A

The clear property lets you control the flow of text around floated elements. You only use the clear property when you are using the float property. Use the clear property to force text to appear beneath a floated element, rather than next to it. Figure 6-20 shows an example of normal text flow around an element.

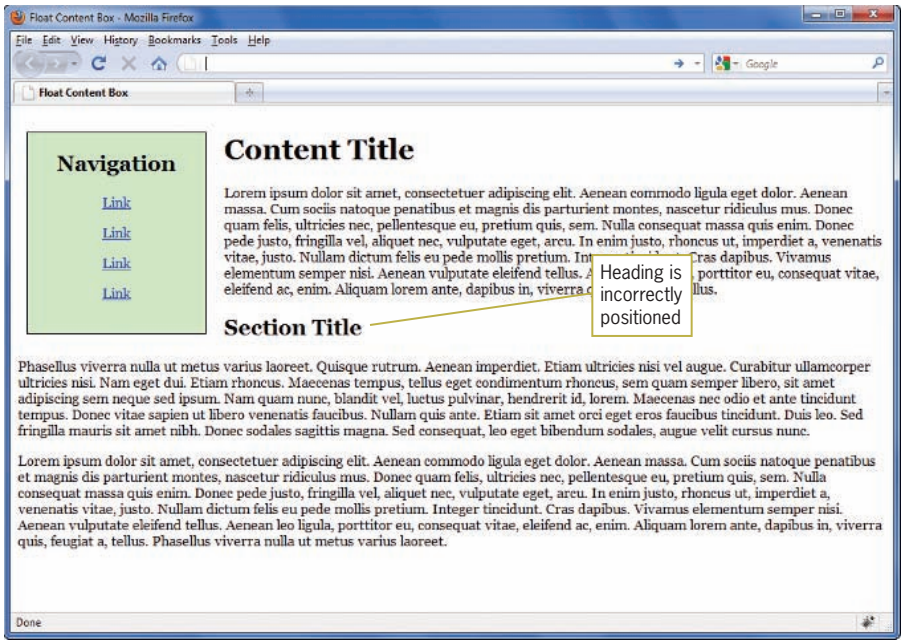


Figure 6-20 Normal text flow around a floating image

This figure shows the left floating content box from the previous example. The text flows down around the image on the right, which is the correct behavior. The second-level heading does not appear in the correct position. It should be positioned beneath the floating content box. To correct this problem, use the clear property. In this instance, the <h2> should be displayed clear of

any left-floating images. Add this style rule directly to the <h2> element with the style attribute as follows:

```
<h2 style="clear: left;">
```

Figure 6-21 shows the result of adding the clear property.

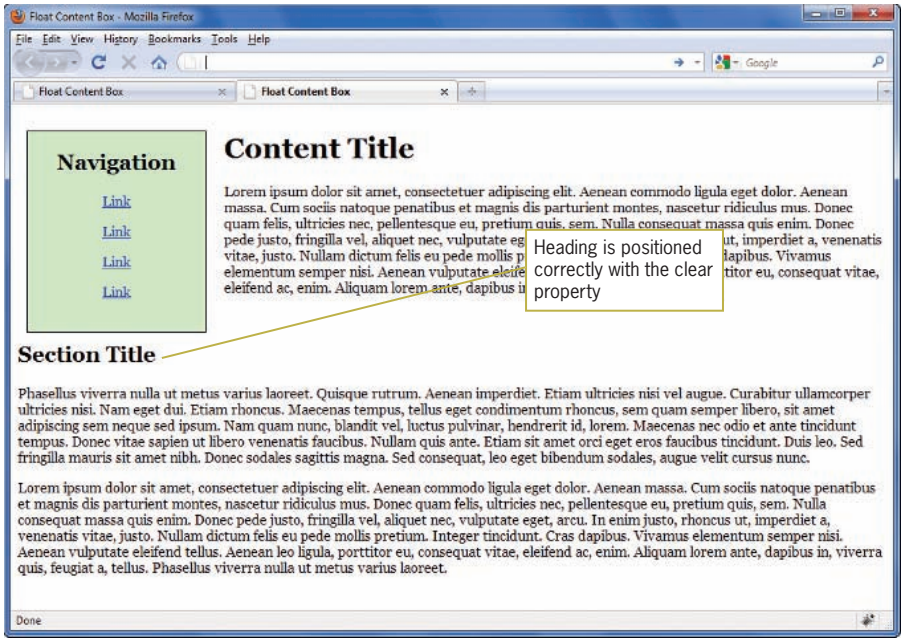


Figure 6-21 Using the clear property

Notice that the clear property lets you clear from either left- or right-floating images using the *left* and *right* values. The *both* value lets you control text flow if you have floating images on both the left and right sides of the text.

Controlling Overflow

overflow property description	
Value:	[visible | hidden | scroll | auto]
Initial:	visible
Applies to:	block-level elements
Inherited:	no
Percentages:	N/A

The overflow property lets you control when content overflows the content box that contains it. This can happen when the content is larger than the area it is designed for, especially when a height property is set for a content element. Figure 6-22 shows text overflow in a content box with fixed width and height.

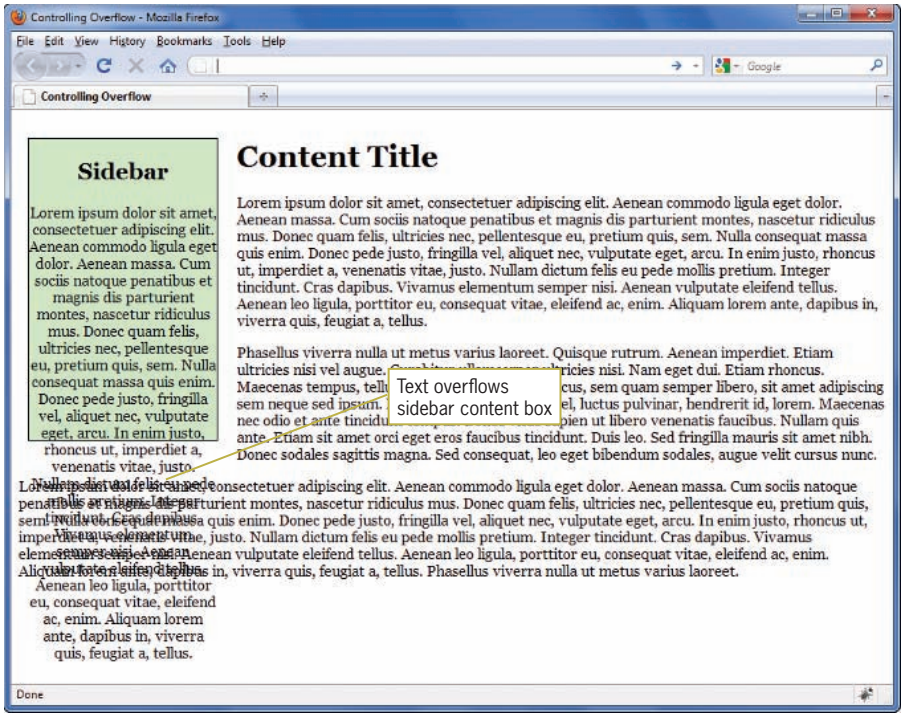


Figure 6-22 Text overflows a box with fixed width and height

To solve this problem, you can add an overflow property to the element. You can choose to show scroll bars or to hide the overflow content. In Figure 6-23, you can see the result of adding the overflow property set to auto, which automatically adds a scroll bar when the content overflows the box.

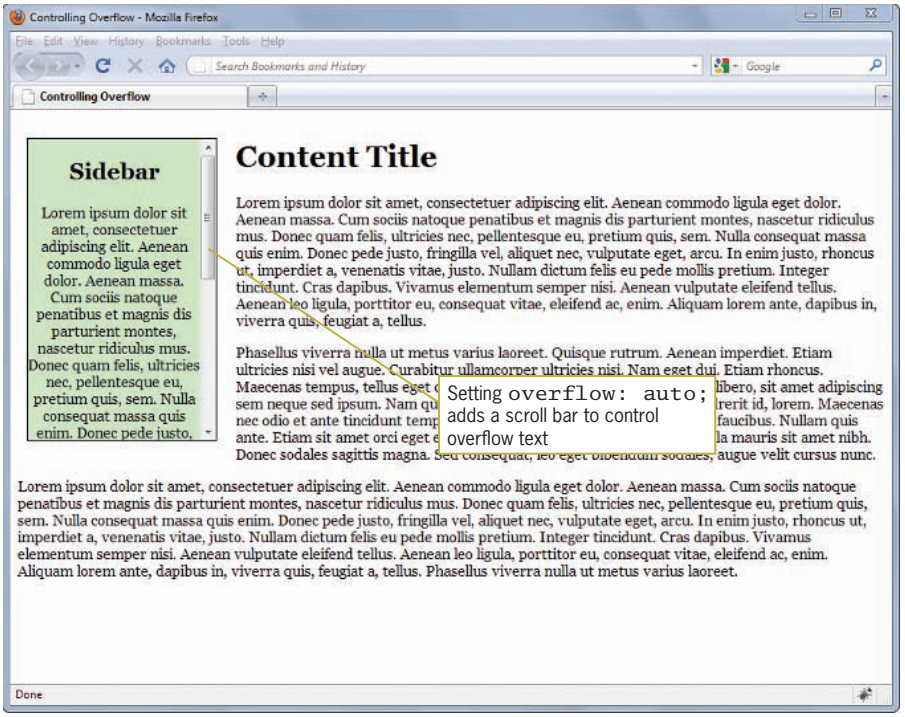


Figure 6-23 Controlling overflow with an auto scrollbar

Creating Box Shadows

box-shadow property description	
Value:	none | <*shadow*> [, <*shadow*>]*
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The box-shadow property lets you add box shadows to an element to create a 3D effect. Figure 6-24 shows this in the Opera browser, which currently is the only browser to support this property.

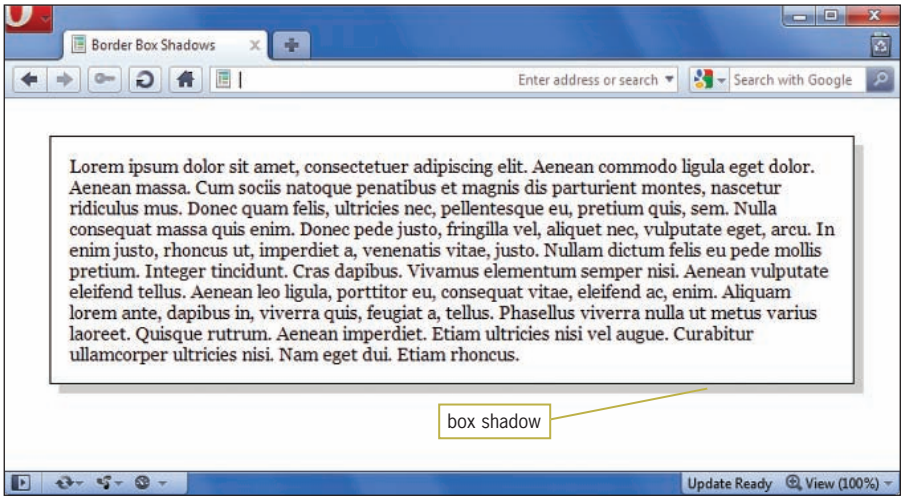


Figure 6-24 Paragraph element with a box shadow

The box-shadow property lets you set both the horizontal and vertical measurement and color for the shadowed edges of the paragraph box. The style rule for this paragraph looks like this:

```
p {
  margin: 2em;
  border: thin solid;
  box-shadow: .5em .5em #ccc;
  padding: 1em;
}
```

The first value is the horizontal offset, and the second is the vertical offset. You can also use negative values to achieve results like the one shown in Figure 6-25.

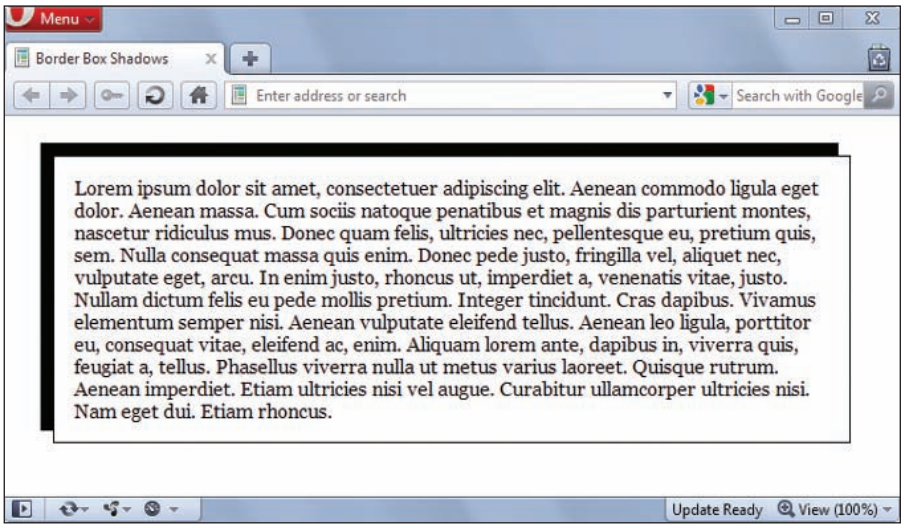


Figure 6-25 Paragraph element with negative box shadow

The style rule for the negative shadow looks like this:

```
p {
  margin: 2em;
  border: thin solid;
  box-shadow: -10px -10px #000;
  padding: 1em;
}
```

Activity: Creating a Simple Page Layout

In the following steps, you have a chance to apply some of the properties you learned about in this chapter to build a simple page layout. You will build on these skills in the next chapter to create more complex page designs. As you work through the steps, refer to Figure 6-29 to see the results you will achieve. New code that you will add is shown in **blue**. Save your file and test your work in the browser as you complete each step.

To apply the box properties:

1. Copy the **box model.html** file from the Chapter06 folder provided with your Data Files to the Chapter06 folder in your work folder. (Create the Chapter06 folder, if necessary.)
2. Open the file **box model.html** in your HTML editor, and save it in your work folder as **box model1.html**.
3. In your browser, open the file **box model1.html**. When you open the file, it looks like Figure 6-26.

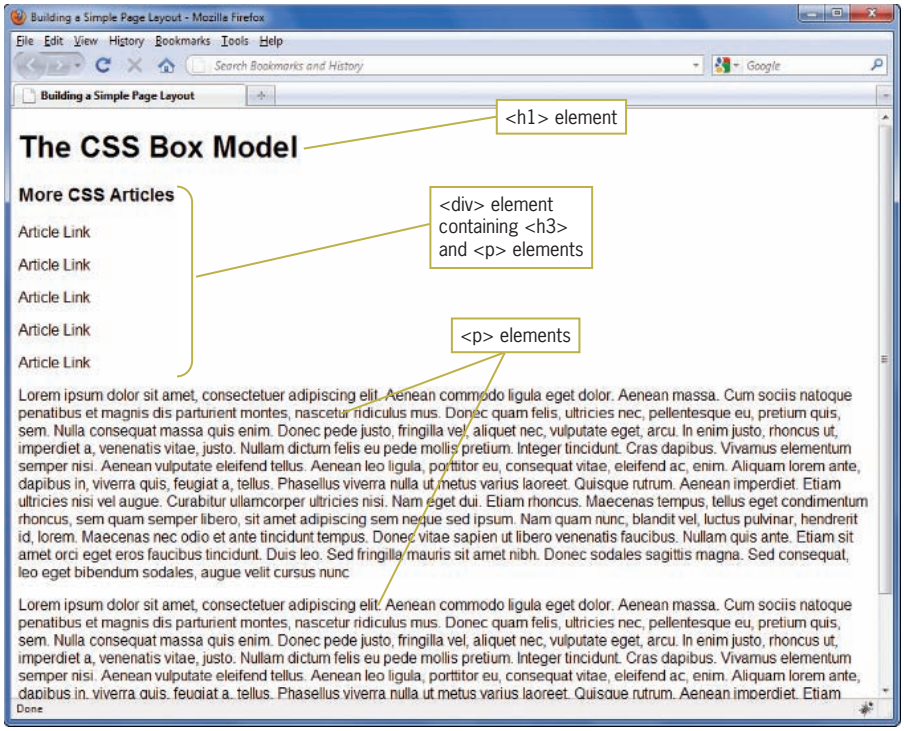


Figure 6-26 Original HTML document

4. Examine the code. Notice the <style> section of the file. It contains one style rule to set the font-family for the page.

```
<style type="text/css">
body {font-family: arial, sans-serif;}
</style>
```

As shown in Figure 6-26, the HTML code in the file consists of:

- <h1> element for the heading
- content container that has an <h3> heading and <p> elements that simulate links
- Two paragraph elements

5. Start by styling the <h1> element with a 20-pixel left and right margin as shown in the following code:

```
h1 {
   margin-left: 20px;
   margin-right: 20px;
}
```

6. Add a border to the left and bottom sides of the <h1> for a distinctive look. Set the width of the border to 4 pixels.

```
h1 {
   margin-left: 20px;
   margin-right: 20px;
   border-bottom: solid 4px black;
   border-left: solid 4px black;
}
```

7. Add 5 pixels of padding to the left and bottom sides of the <h1> to offset the text from the border you just created.

```
h1 {
   margin-left: 20px;
   margin-right: 20px;
   border-bottom: solid 4px black;
   border-left: solid 4px black;
   padding-bottom: 5px;
   padding-left: 5px;
}
```

The finished <h1> element looks like Figure 6-27.

Figure 6-27 Completed <h1> element

8. Now you will create the floating content box that contains the "More CSS Articles" heading and links. Write a style rule that uses an id of *sidebar*. Add a width of 200 pixels and a solid 1-pixel border.

```
#sidebar {
   width: 200px;
   border: 1px solid black;
}
```

9. In the <body> section of the document, find the element that contains the "More CSS Articles" and add the id attribute to apply the sidebar style as shown:

```
<div id="sidebar">
<h3>More CSS Articles</h3>

<p>Article Link</p>
<p>Article Link</p>
<p>Article Link</p>
```

```
<p>Article Link</p>
<p>Article Link</p>
</div>
```

10. Float the division to the left side of the page. Add a left margin of 20 pixels to align the box with the heading and a right margin of 20 pixels to offset the paragraph text from the box.

```
#sidebar {
   width: 200px;
   border: 1px solid black;
   float: left;
   margin-left: 20px;
   margin-right: 20px;
}
```

11. Finish the content box by centering the text and adding a background color (#cff).

```
#sidebar {
   width: 200px;
   border: 1px solid black;
   float: left;
   margin-left: 20px;
   margin-right: 20px;
   text-align: center;
   background: #cff;
}
```

Figure 6-28 shows the completed floating sidebar content box. Notice that the paragraph text is very close to the left and right sides of the browser window.

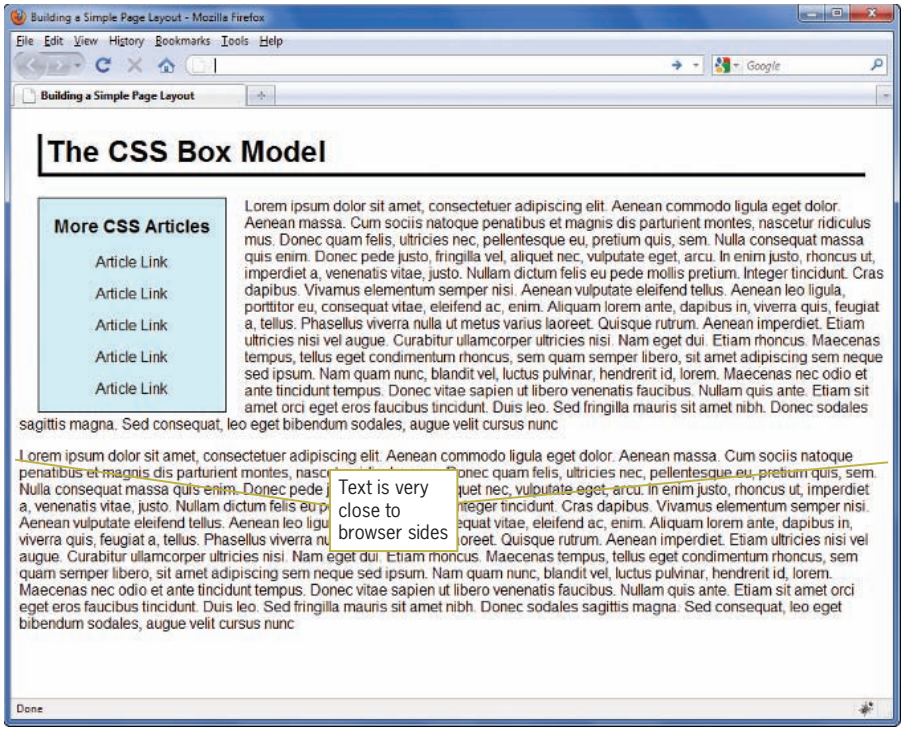


Figure 6-28 Floating content box

12. Finish the design by aligning the text properly using left and right margin properties. Write a rule that selects the <p> elements and sets the left margin to 20 pixels and the right to 40 pixels.

```
p {
   margin-left: 20px;
   margin-right: 40px;
}
```

Figure 6-29 shows the completed Web page design.

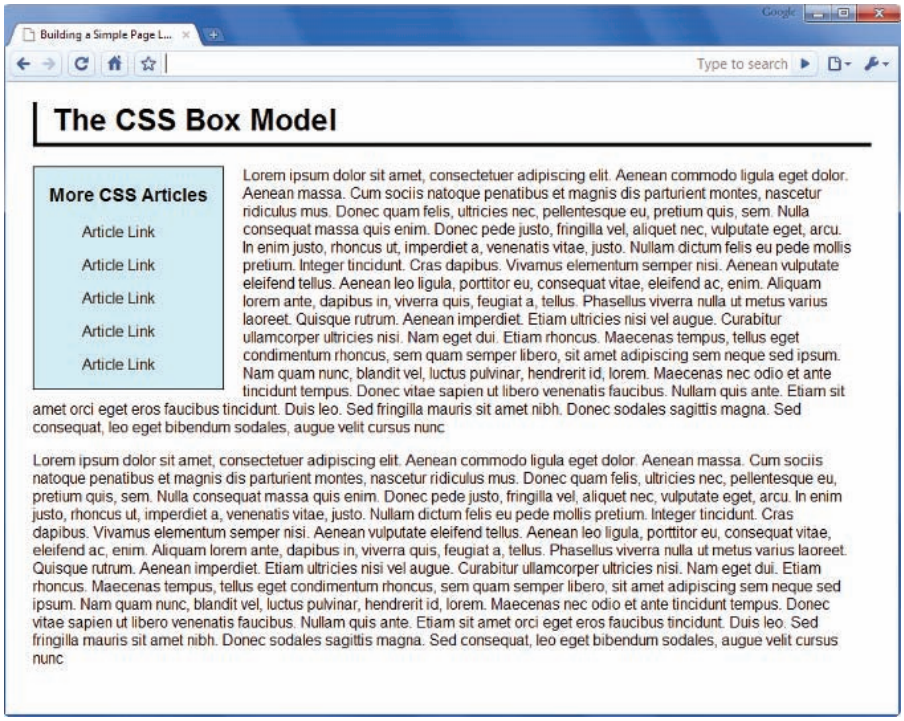


Figure 6-29 Completed Web page design

The finished style sheet code:

```
<style type="text/css">

body {font-family: arial, sans-serif;}

h1 {
   margin-left: 20px;
   margin-right: 20px;
   border-bottom: solid 4px black;
   border-left: solid 4px black;
   padding-bottom: 5px;
   padding-left: 5px;
}

#sidebar {
   width: 200px;
   border: 1px solid black;
   float: left;
   margin-left: 20px;
   margin-right: 20px;
   text-align: center;
   background: #cff;
}
```

```
p {
   margin-left: 20px;
   margin-right: 40px;
}

</style>
```


Chapter Summary

In this chapter you learned about the concepts of the CSS box and visual formatting models. You saw how the margin, padding, and border properties let you control the space around block-level elements on a Web page. By using these properties judiciously, you can enhance the legibility of your content. You also learned how the page layout box properties create boxes and columns for content, wrap text, and float images.

- The CSS box model lets you control spacing around the element content.
- You can state values of margin, border, and padding for all four sides of the box or individual sides.
- To build scalable Web pages, choose relative length units such as ems or percentages.
- To build fixed pages, choose pixel measurements.
- The browser collapses vertical margins to ensure even spacing between elements.
- Margins are transparent, showing the color of the containing element's background color. Padding takes on the color of the element to which it belongs.
- The border properties let you add borders to all individual sides or all four sides of an element. The three border characteristics are style, color, and width. Style must be stated to make the border appear.
- The page layout box properties let you create floating content boxes and wrap text around images.
- Remember to use margin, border, and padding properties to enhance the legibility of your content.

Key Terms

border properties—CSS properties that let you control the appearance of borders around elements.

box model—A CSS element that describes the rectangular boxes containing content on a Web page.

containing box—The containing rectangle, or parent element, of any child element. The absolute containing element is the window area of the browser. All other elements are displayed within this containing box, which equates to the <body> element of an HTML document. Within <body>, elements such as or <p> are parents, or containing boxes, for their child elements.

CSS visual formatting model—A model that describes how CSS element content boxes should be displayed by the browser.

margin properties—CSS properties that let you control the margin area of the box model.

padding properties—CSS properties that let you control the padding area in the box model.

page layout box properties—CSS properties that let you control the dimensions and position of content boxes.

Review Questions and Exercises

1. What are the three space areas in the box model?

2. Which space area is transparent?

3. What does the visual formatting model describe?

4. What is the visual formatting model based on?

5. What are percentage measurement values based on?

6. What are the preferred length units for margins and padding?

7. In the following rule, what is the size of the vertical margins between paragraphs?

```
p {margin-top: 15px; margin-bottom: 10px;}
```

8. Where is the padding area located?

9. What are the five most common border properties?

10. What is the default border style?

11. What is the default border weight?

12. What is the default border color?

13. What does the float property let you do?

14. What does the clear element let you do?

15. Write a style rule for a <p> element that sets margins to 2em, padding to 1em, and a black, solid 1-pixel border.

16. Write a style rule for an <h1> element that sets top and bottom padding to .5em with a dashed thin red border on the bottom.

17. Write a style rule for a <p> element that creates left and right padding of 1em, a left margin of 30 pixels, and a left black medium double border.

Hands-On Projects

1. Browse the Web and choose a site that you feel exhibits good handling of white space to increase the legibility of the content. Write a short design critique of why the white space works effectively.

2. Browse the Web and choose a site that can benefit from the box properties available in CSS. Print a copy of the page and indicate where you would change the spacing and border properties. Write a short essay that describes the changes you want to achieve and how they would increase the legibility of the page content.

3. In this project, you will create a floating text box.

   a. Copy the **floatactivity.html** file from the Chapter06 folder provided with your Data Files to the Chapter06 folder in your work folder. (Create the Chapter06 folder, if necessary.)

b. Open the file **floatactivity.html** in your HTML editor, and save it in your work folder as **floatactivity1.html**.

c. In your browser, open the file **floatactivity1.html**. When you open the file, it looks like Figure 6-30.

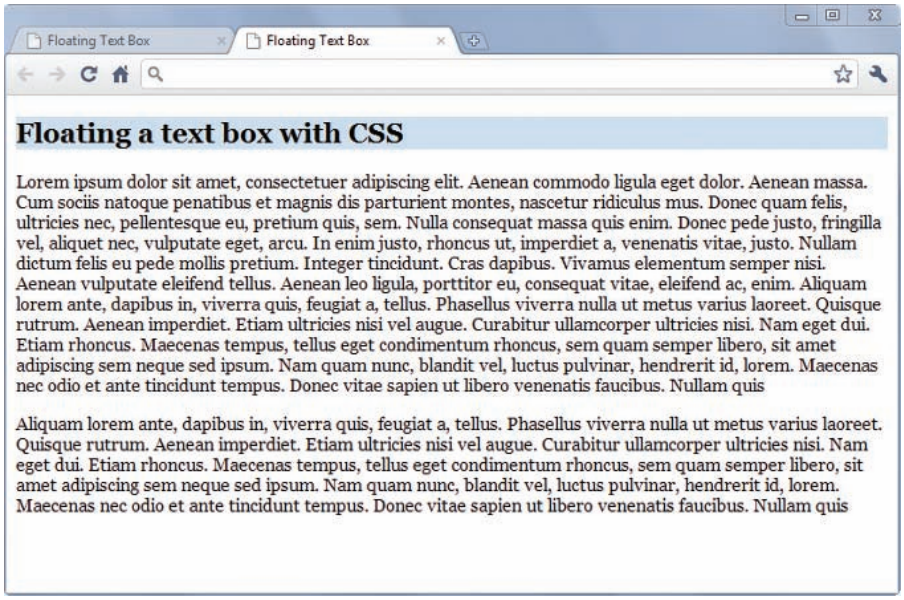


Figure 6-30 Original HTML file for Project 3

d. Examine the page code. Notice that an existing style rule sets a background-color for a floatbox class, as shown in the following code fragment:

```
.floatbox {background-color: #ccddee;}
```

e. This class is applied to the first <p> element in the document, as shown in Figure 6-30. Your goal is to use a variety of box properties to create a finished page that looks like Figure 6-31.

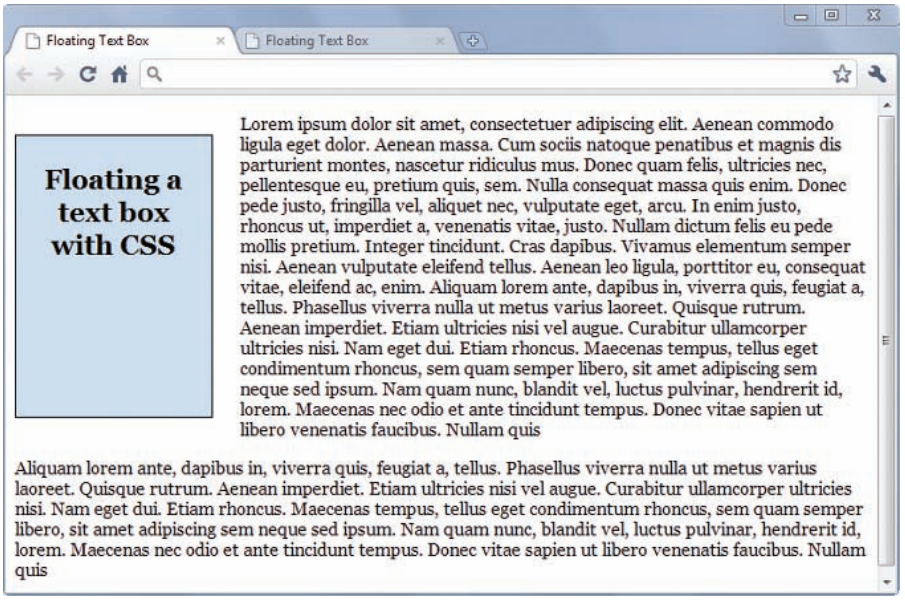


Figure 6-31 Completed HTML file for Project 3

f. Use the following properties to create the finished floating text box:

- width
- height
- float
- padding
- margin-right
- border
- text-align

Experiment with the different properties until you achieve results that look as close to the finished page as possible.

4. In this project, you will have a chance to test the border properties. Save and view the file in your browser after completing each step.

   a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains heading and paragraph elements. Save the file in your Chapter06 folder as **borders.html**.

b. Add a <style> element to the <head> section as shown in the following code:

```
<head>
<title>CSS Test Document</title>
<style type="text/css">
</style>
</head>
```

c. Experiment with the different border styles. Start by applying any of the following style rules to your document's elements:

```
h1 {border: solid 1px black;}
h2 {border-top: solid 1px; border-bottom: solid 3px;}
p {border-left: double red; border-right: solid 1px;}
```

d. Experiment with adding padding properties to your style rules to offset the borders from the text. The following style rules have sample padding properties to try:

```
h1 {border: solid 1px black; padding: 20px;}
h2 {border-top: solid 1px; border-bottom: solid 3px;
   padding-top: 15px; padding-bottom: 30px;}
p {border-left: double red; border-right: solid 1px;
   padding-left: 30px; padding-right: 20px;}
```

e. Continue to experiment with the border and padding properties. Try adding color and margin properties to see how the elements are displayed.

Individual Case Project

Create the box model conventions for your Web site. Build on the typographic classes you created in Chapter 5. Think about the different spacing requirements for your content and decide how the legibility can be enhanced using the box properties. Add this information to the type specification HTML page that shows examples of the different typefaces and sizes and how they will be used. Decide on margins, padding, and borders and select the elements that will benefit from their use. Create before and after sample HTML pages that reflect the enhanced design.

Team Case Project

Work with your team to decide on the box model conventions for your project Web site. These include any spacing specifications for your content that will increase legibility and clarity.

You may need to create sample mock-up HTML pages to present your ideas on these characteristics to your team members. Decide on margins, borders, and padding and how you will use these in your Web site. Think about creating floating boxes within sections for text highlights, how to standardize the look of different sections of your site with increased white space, and how borders can help to emphasize headings and columns.

After you have reached a general consensus, go back to work on your page template. Create more finished mock-ups of your page design. Trade the page layout examples with your team members to reach consensus.

CHAPTER 7

Page Layouts

When you complete this chapter, you will be able to:

- Understand the normal flow of elements
- Use the division element to create content containers
- Create floating layouts
- Build a flexible page layout
- Build a fixed page layout

In a standard HTML document, the default positioning of elements is generally top to bottom and left to right. In the past, Web designers used tables to create multiple column layouts and gain more control of their page designs. HTML tables are not intended for page layout and are no longer in favor, although they still exist on many Web pages.

Modern Web designs are built using the CSS layout capabilities. As you saw in Chapter 6, you can use floats to position content elements on a Web page and move them out of the normal flow of elements. In this chapter, you will learn how to expand on this concept by using floats to create multicolumn Web pages that can either be flexible based on the browser size and screen resolution, or fixed to a definite width. You will see how to resolve common float problems, and you will get a chance to build both a flexible and a fixed page layout.

Understanding the Normal Flow of Elements

By default, the browser normally displays elements on the page one after the other, depending on whether the elements are block-level or inline elements. Some elements float to the left or right of other elements on the page, as you saw in Chapter 6. Element position can be affected by margin or padding properties, but generally the browser lays out element boxes from top to bottom and left to right until all elements that make up the Web page have been displayed.

In the normal flow for block-level elements, boxes are laid out vertically one after the other, beginning at the top of the containing block. Each box horizontally fills the browser window. The space between boxes is determined by the margin settings. The **normal flow** determines the sequence of element display with which you are familiar in standard HTML. For an example of normal flow, examine the following HTML code and the resulting normal flow diagram in Figure 7-1:

```
<body>
   <h1>The Document Title</h1>
   <p>Lorem ipsum...</p>
   <p>Duis autem...</p>
   <p>Ut wisi enim...</p>
</body>
```

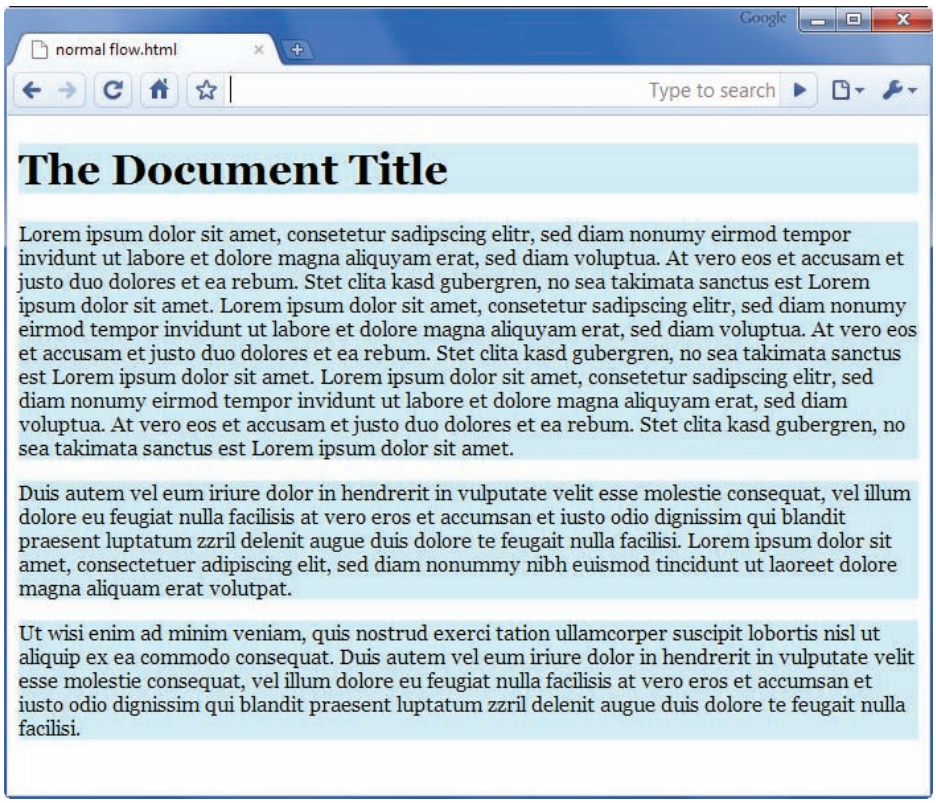


Figure 7-1 Block-level element normal flow

The <body> element is the containing element for the content section of the Web page. The elements within the <body> element are displayed exactly in the order they appear in the code from top to bottom, which in this example is an <h1> followed by three <p> elements. Elements do not appear next to each other unless they are floated (see Chapter 6) or have a display type of *inline*.

In the normal flow for inline elements, boxes are laid out horizontally, beginning at the top left of the containing block. The inline boxes comprise the lines of text within, for example, a <p> element. The browser flows the text into the inline boxes, wrapping the text to the next line box as determined by the constraints of the containing box or the browser window size.

When you **float**, or position an element, you take it out of the normal flow. Other elements that are not floated or positioned will still follow the normal flow, so you should check the results frequently as you are designing your layout using floats. Figure 7-2 shows a floated element on the left side of the page. Notice that

the two other nonfloating elements remain in the normal flow and still span the width of the browser window, even extending behind the floating elements. As you can see in the example, the text in the normal flow boxes appears in the correct position to the right of the floated element. This behavior allows text to wrap around floated elements such as images, which is a basic advantage of floats. However, when you start to use floats to build page layouts, the behavior of floats can cause problems. As you gain more experience working with floating layouts, you will be able to anticipate and correct problems with floats as you code your page designs.

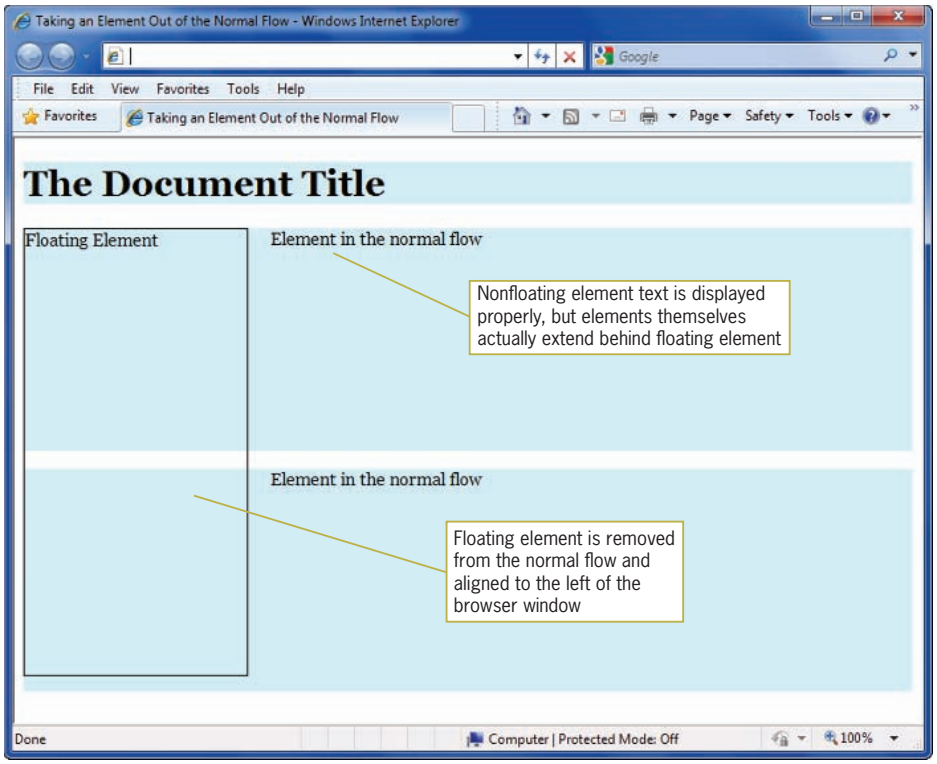


Figure 7-2 Floating element is removed from the normal flow

Using the Division Element to Create Content Containers

The division element is your primary tool for creating sections of content in your Web page designs. Using the box properties you learned about in Chapter 6, you can create divisions that are any

shape you need to contain and segregate sections of content. You can create vertical columns containing content and control the white space between and within the columns. You can nest divisions within divisions and create interesting content presentations. Finally, you can create a division element to contain an entire Web page, often called a **wrapper**, to center a Web page within the browser window, regardless of screen resolution.

Figure 7-3 shows a Web page created with multiple divisions. The wrapper division, outlined in red, contains all of the content of the Web page. The wrapper has three child division elements; header, navigation, and article. Each of these divisions will contain page content.

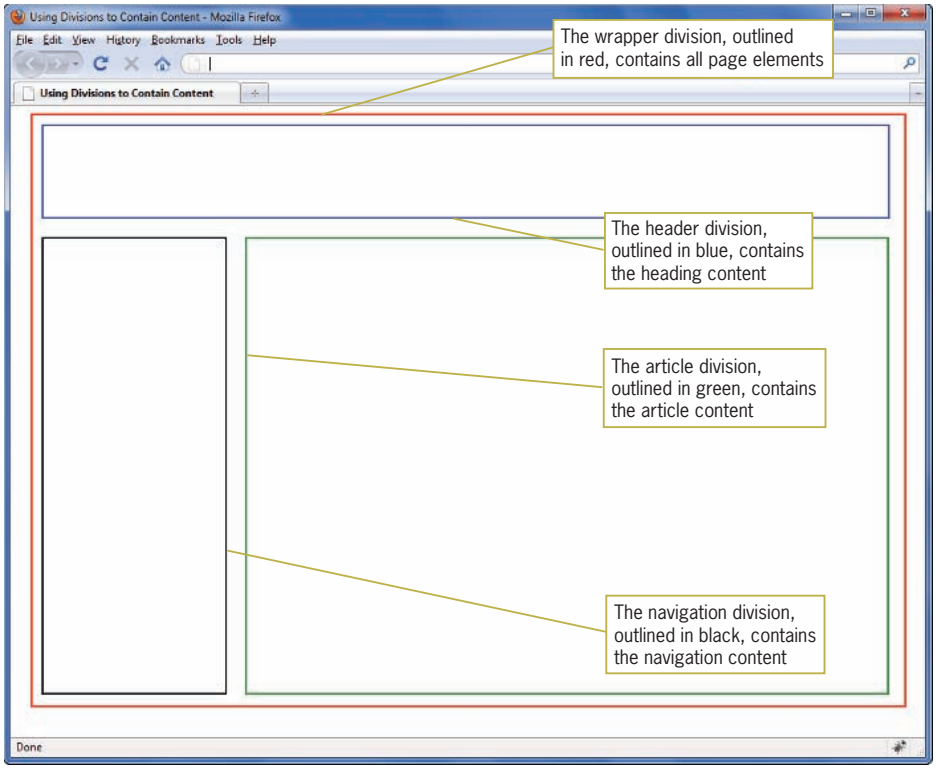


Figure 7-3 Web page with multiple content divisions

In Figure 7-4, you can see these same divisions with sample content. The banner division contains an <h1> element. The navigation division contains an <h2> heading and links. The main division contains an <h2> heading, an image and paragraph elements. Various margin and padding settings offset the content

from the sides of the container elements. The wrapper element holds all the pieces together, and lets the page be centered in the browser window.

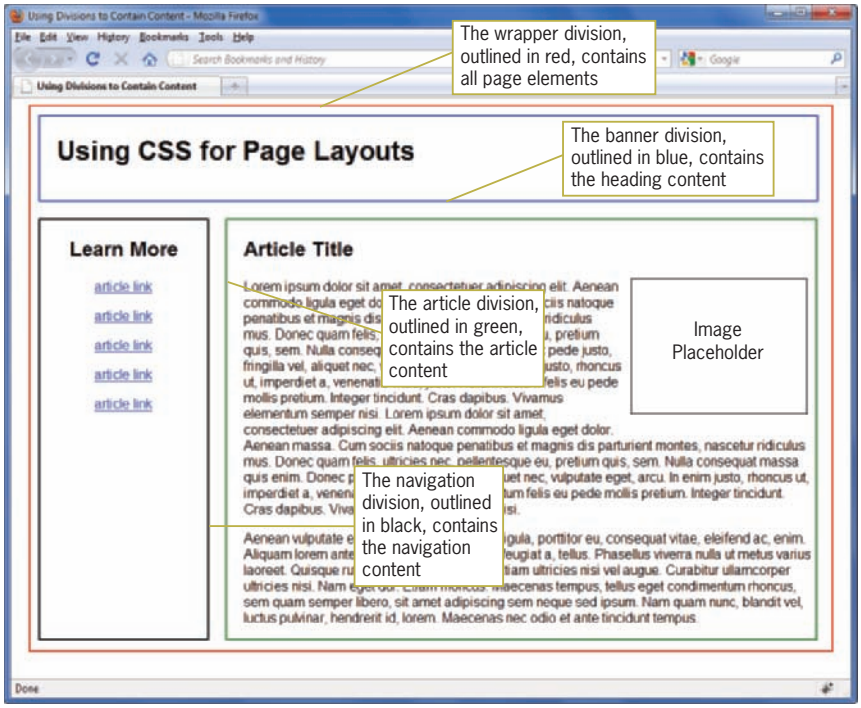


Figure 7-4 Web page with multiple content divisions, filled with content

Creating Floating Layouts

The float property lets you build columnar layouts by floating content elements to either the right or left side of the browser window. A typical Web page design can contain both floating and nonfloating elements. For example, Figure 7-5 shows a Web page layout with a header, three columns of content, and a footer. The three columns are floating divisions, while the header and footer are nonfloating. The nav and article columns are floating to the left, while the sidebar is floating to the right. All of the page elements are separated from each other with margin settings to provide gutters between the columns.

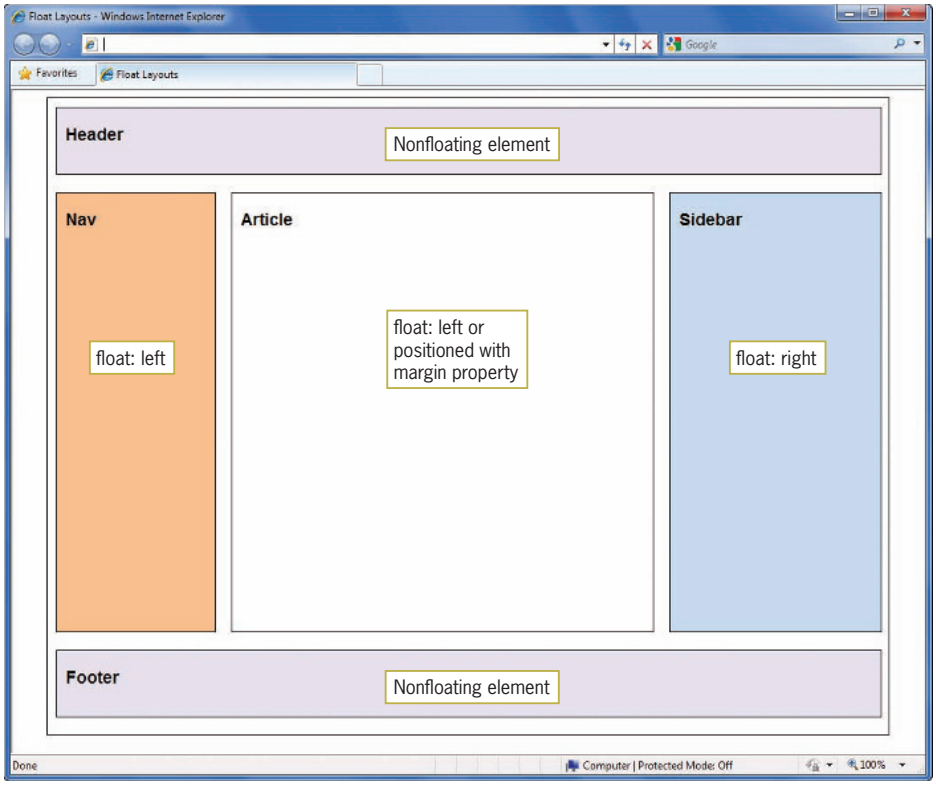


Figure 7-5 Floating and nonfloating elements make up a Web page layout

Building floating layouts requires that you choose a method for containing the floating elements. Floats are designed to extend outside of their containing element. This is because the original concept of floating was to allow text to wrap around images, as you saw earlier in this chapter. When you start to build floating layouts, you will often see that the floating elements extend beyond their containing elements, which will result in a "broken" layout as illustrated in Figure 7-6.

Floating elements must always have a specified width or they will expand to the size of the browser window.

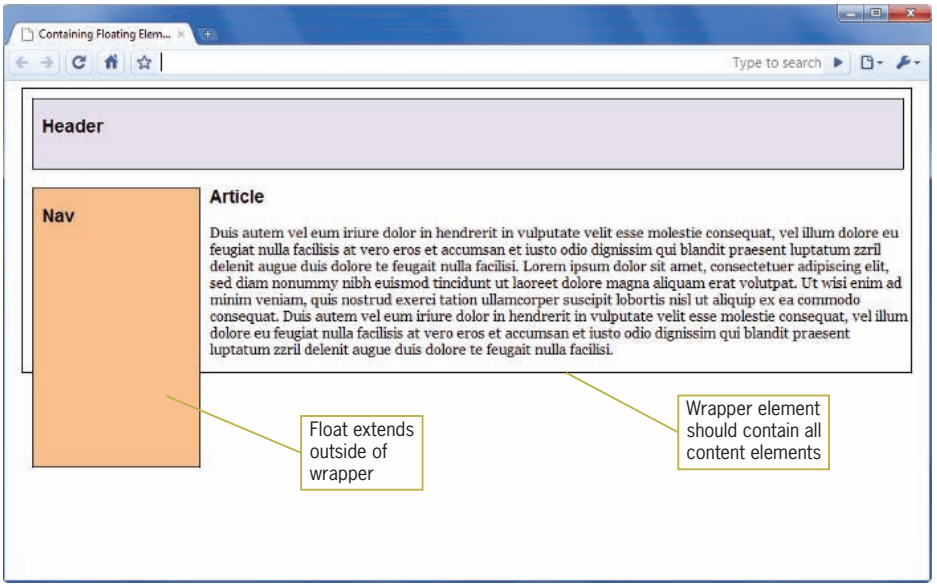


Figure 7-6 Floating element extends outside of containing element

There are two methods you can use to fix this problem.

Solution 1: Using a Normal Flow Element

If you have multiple columns, at least one needs to be nonfloating (in the normal flow), and positioned with margin properties, as shown in Figure 7-7.

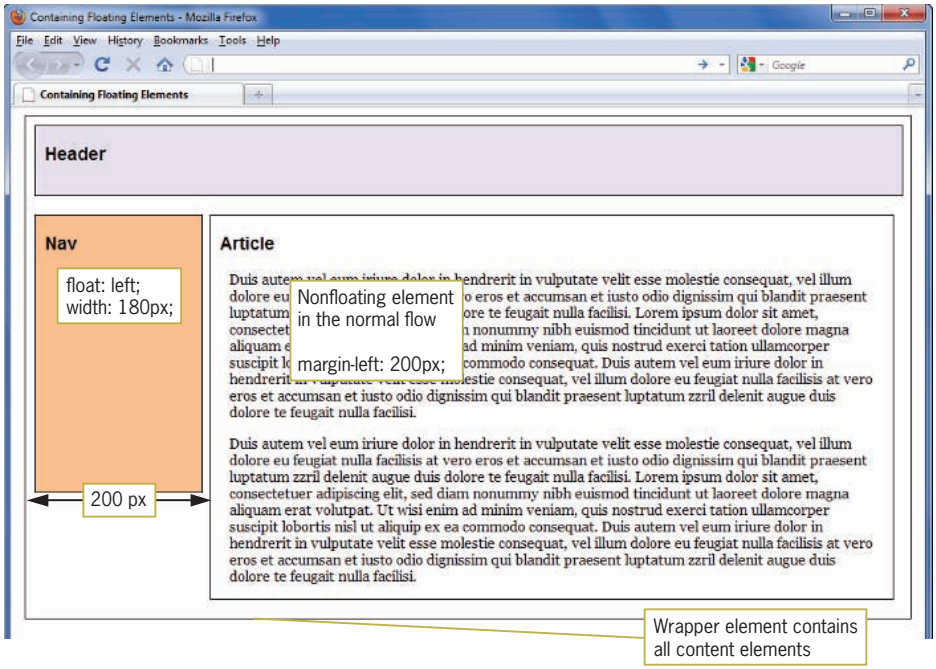


Figure 7-7 Using a normal flow element to contain floats

In this example, the *article* element is nonfloating and has a 200-pixel left margin to position the element to the right of the floating *nav* element. The style rule for both the *nav* and the *article* elements looks like this:

```
#nav {
   width: 180px;
   height: 300px;
   float: left;
   border: solid thin black;
   margin-top: 20px;
   margin-left: 10px;
   margin-right: 10px;
   background-color: #fabf8f;
}

#article {
   width: 740px;
   margin-left: 200px;
   border: solid thin black;
   background-color: #fff;
   margin-top: 20px;
   margin-bottom: 20px;
}
```

Solution 2: Using the Clear Property

If you use a nonfloating footer element (in the normal flow), with the clear property set to *both*, the containing wrapper will extend to contain all elements, as shown in Figure 7-8.

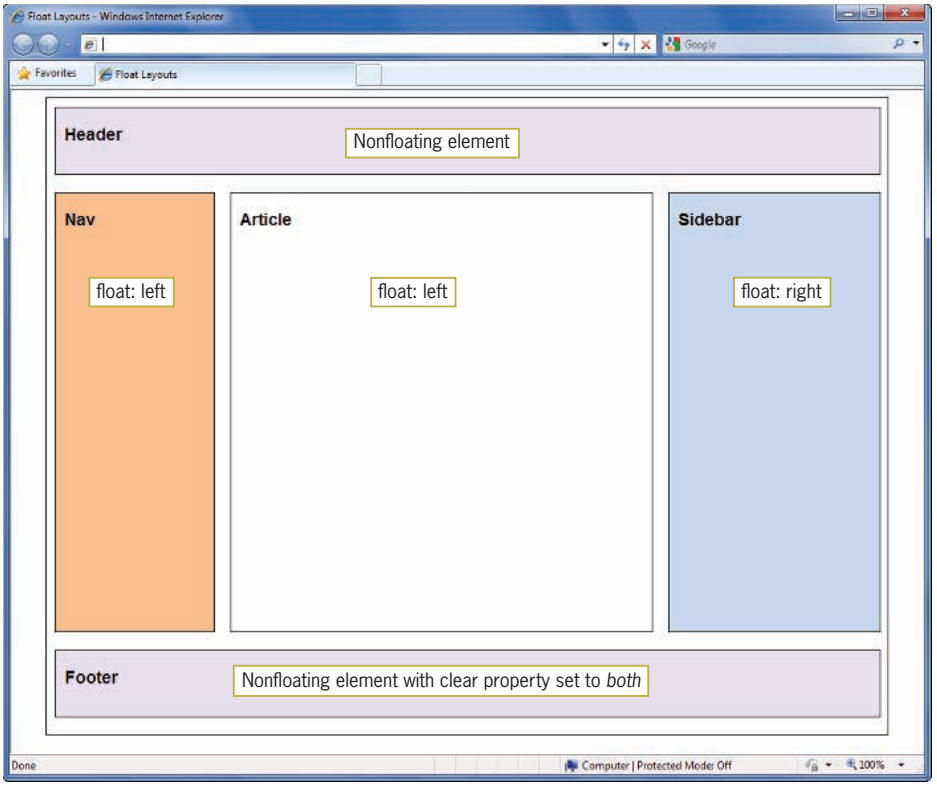


Figure 7-8 Using the clear property to contain floats

In this example, the footer element is nonfloating and has the clear property set to *both*. This forces the wrapper to extend beyond the footer property and contain all of the content elements on the page. The style rule for the footer elements looks like this:

```
#footer {
   width: 940px;
   height: 75px;
   margin-left: 10px;
   clear: both;
   border: solid thin black;
   background-color: #e5dfec;
}
```

Because some type of footer is a consistent design feature in most Web sites, this second solution works very well. Your footer can be as simple as a horizontal rule <hr> element or a graphic contained with a footer division.

Floating Elements Within Floats

Using floating elements gives you a wide variety of options for building interesting page layouts. For example, you can float elements within floating elements, as shown in Figure 7-9.

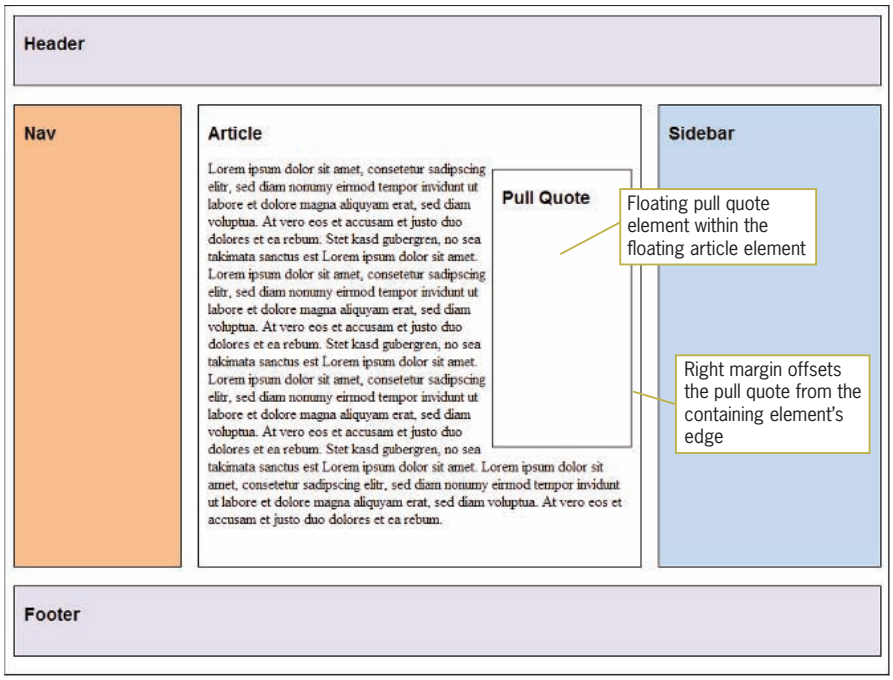


Figure 7-9 Floating element within a float

In this example, a floating element within the article element contains a pull quote, a quotation or excerpt from the main article that would be used to draw the reader's attention to the article. This element floats right within the article element, and is offset from the border of the article element with a right margin. An image element can also be floated using this same method.

When you are floating an element within another element, the order of the elements is important. In the example in Figure 7-10, the floating pull quote element follows the heading element and precedes the paragraph content. If this order is not followed, the

pull quote would appear at the bottom of the article rather than in its correct position. Figure 7-10 shows the order of the elements within the article element.

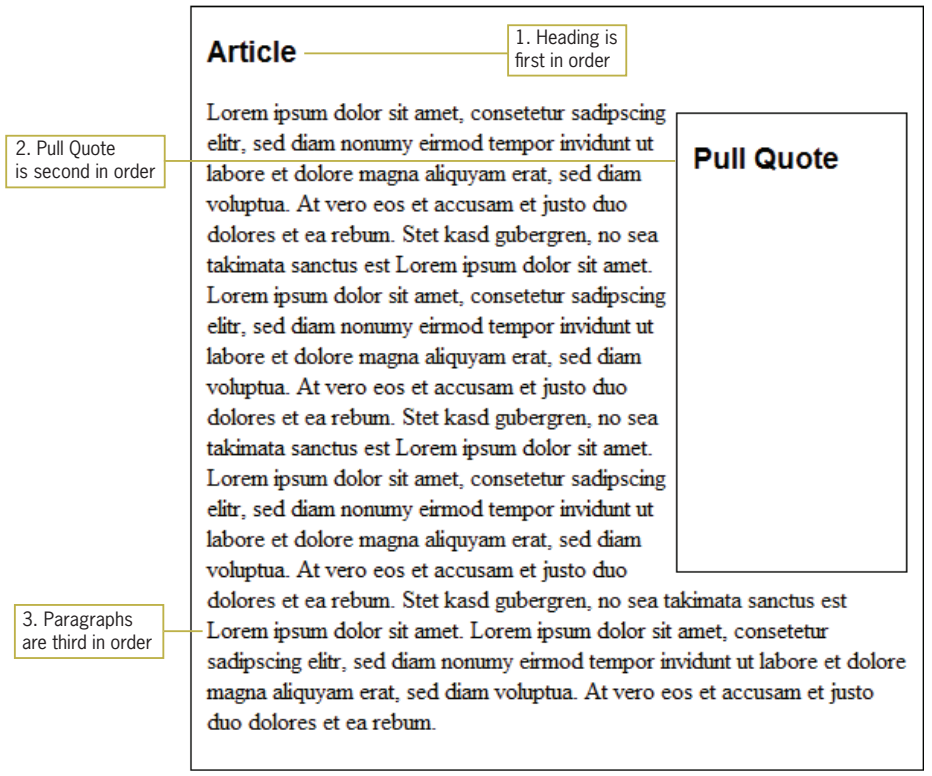


Figure 7-10 Float order is important

In contrast, Figure 7-11 shows what happens to the layout when the correct float order is not maintained. In this example, the order of the pull quote element and the paragraphs has been switched, with unintended results. The pull quote is pushed out of its container element because the paragraph elements take up all of the room in the article element.

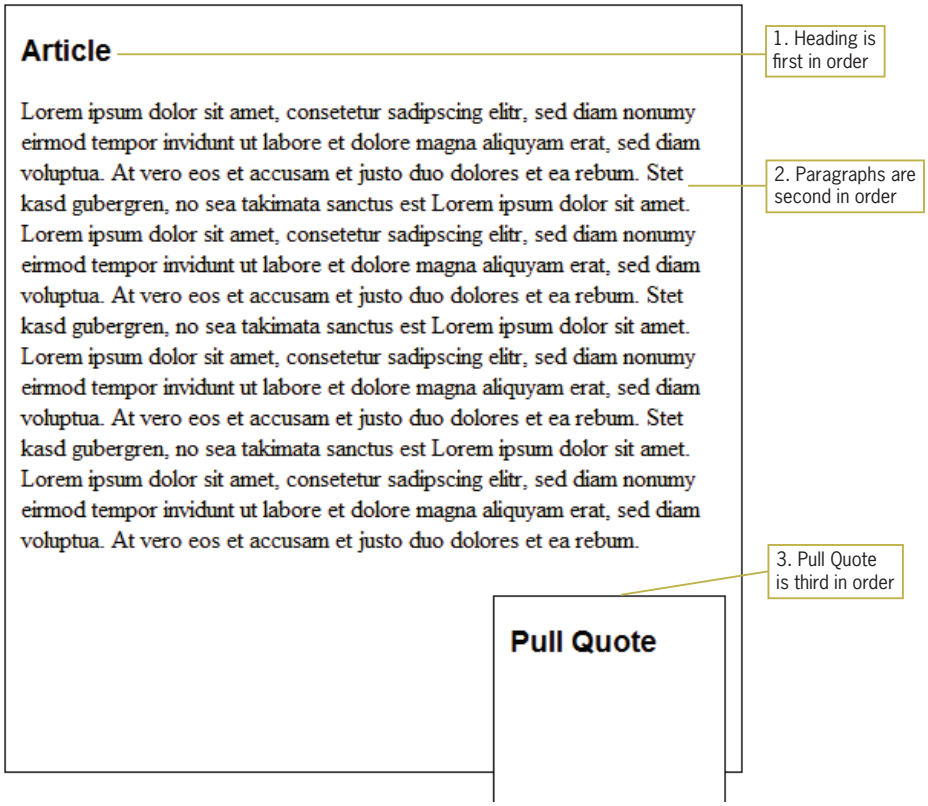


Figure 7-11 Incorrect float order causes layout problems

Fixing Column Drops

Column drops occur when the total width of the columnar elements in a page layout exceeds the width of their containing element. The width of a box element includes the total of its width value plus any left or right padding, border, and margins. Figure 7-12 shows an example of column drop caused by the total width of the contained columns being greater than the width of the containing wrapper element.

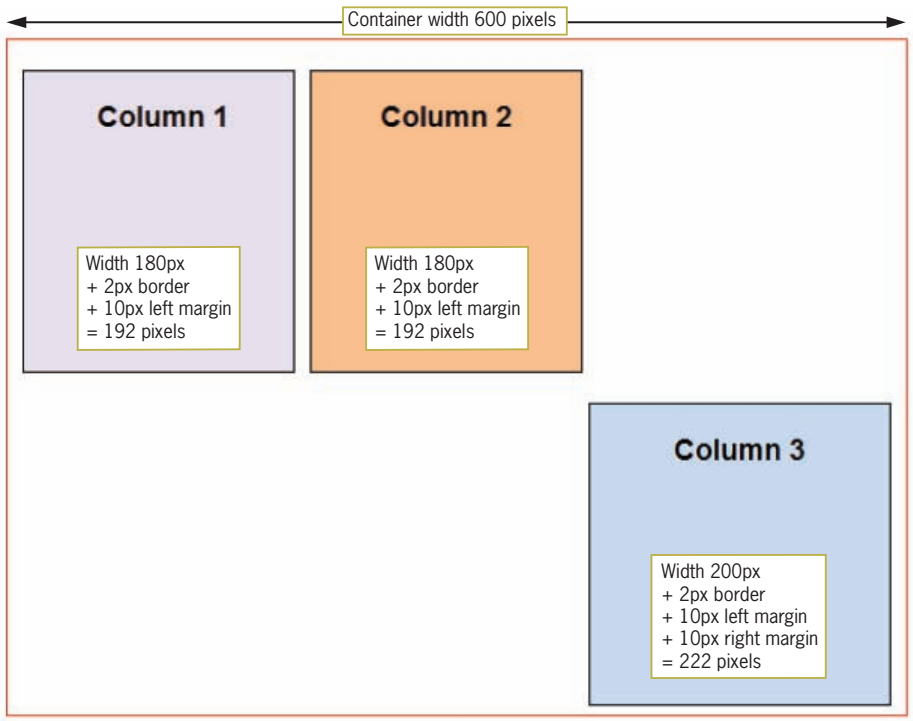
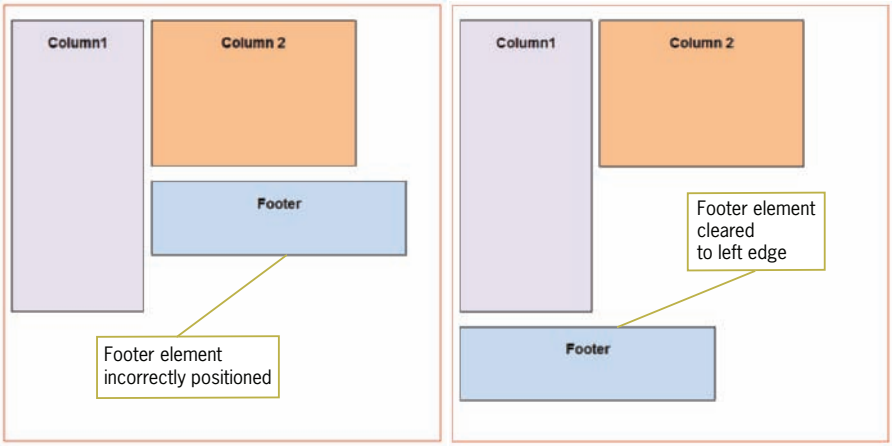


Figure 7-12 Column drop caused by combined element width being greater than container width

If you total the width of the three elements (192 + 192 + 222 = 606), the sum is greater than 600 pixels, the width of the containing element, forcing Column 3 to drop below the other columns because the layout does not provide enough horizontal width. To solve this problem, the width of Column 3 can be reduced to 180 pixels, reducing the overall width of the columns to 586 pixels.

Clearing Problem Floats

When you are designing float-based layouts, floats occasionally do not appear exactly where you want them to appear. The clear property can help you solve this problem. For example, in Figure 7-13a, the footer element floats left and should appear below Column 1, but instead it floats in order after Column 2. To move it down the page to its correct position, add the clear property set to a value of left to move the footer down the page and align it with the next clear left edge of the browser window, as shown in Figure 7-13b.



a. b.

Figure 7-13 Problem floats fixed with the clear property

Building a Flexible Page Layout

Flexible layouts, sometimes known as liquid layouts, adapt to the size of the browser window. Flexible layouts shift as the user resizes the window, wrapping text or adding white space as necessary. In Figure 7-14, the flexible layout contains three elements.

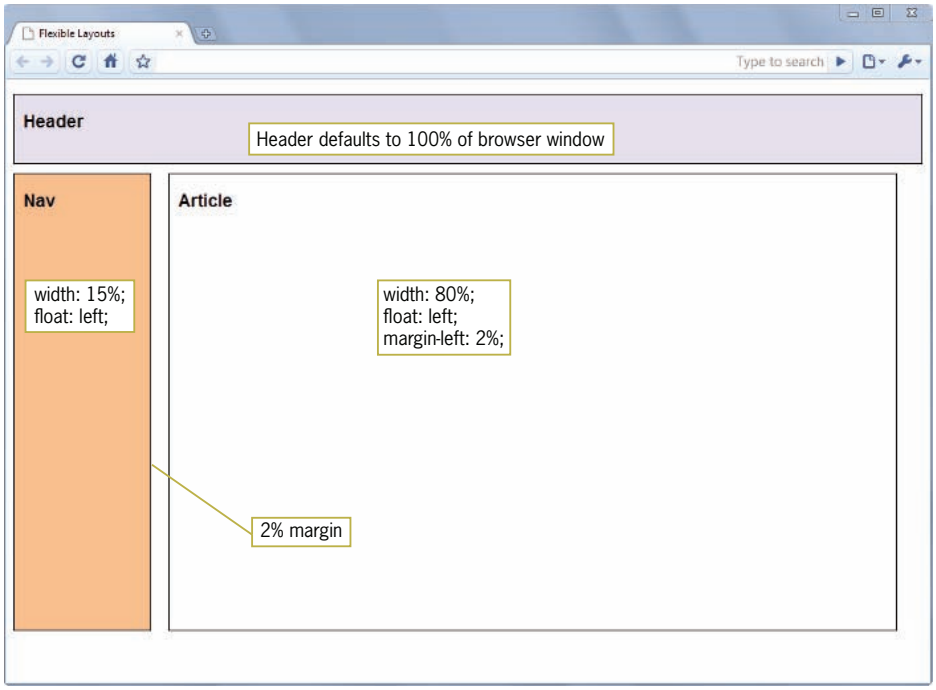


Figure 7-14 Flexible layout

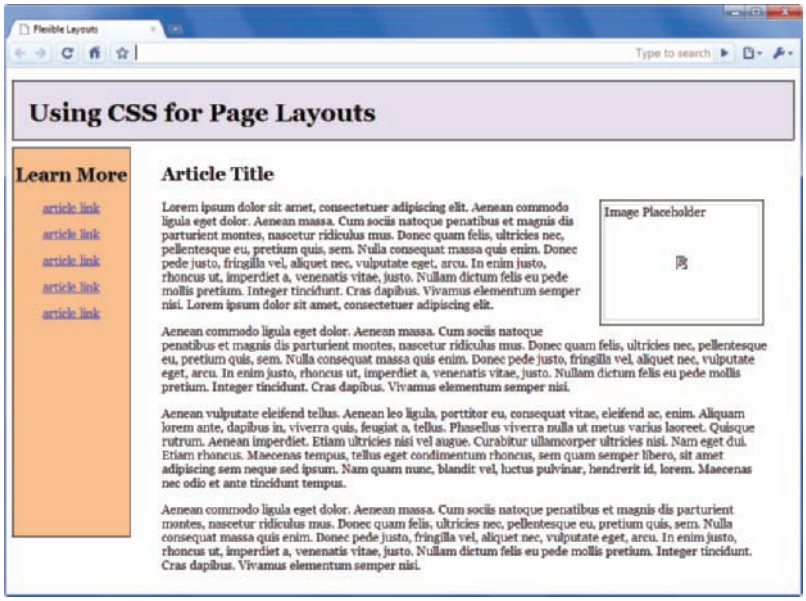
The header element has no width set, so it defaults to the width of the browser window as any normal flow element would. Below the header element are two floating elements, nav and article. These elements have flexible widths set as well. The style rules for these three elements follows:

```
#header {
   height: 75px;
   margin-top: 1em;
   margin-bottom: 10px;
   border: solid thin black;
   background-color: #e5dfec;
}

#nav {
   width: 15%;
   height: 500px;
   float: left;
   border: solid thin black;
   background-color: #fabf8f;
}

#article {
   width: 80%;
   height: 500px;
   float: left;
   margin-left: 2%;
   border: solid thin black;
   background-color: #fff;
}
```

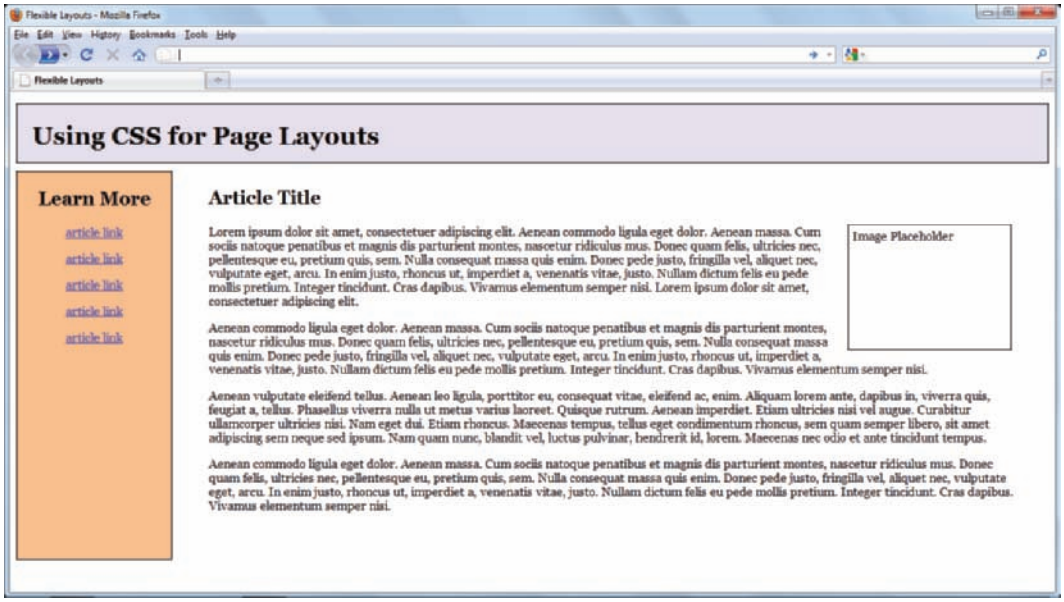
In the code example for Figure 7-14, note that the height property is used in both the nav and article elements. You normally want to avoid using height on elements that contain varying lengths of content, as too much content will overflow a fixed height.

Notice that the two widths for the floating elements do not equal 100% because the borders and margin contribute to the width of each element. Setting the values to equal 100% would overflow the layout window. The nav and article elements both float left. The article element has a 2% left margin, meaning that the margin changes size as the layout resizes to display a consistent gutter between the floating elements.

Flexible layouts offer the advantage of adapting to the user's browser size. From a design viewpoint, this can be less desirable because of the wide range of monitor sizes and resolutions. With a flexible layout, your content has to adapt and look good at wide range of layout sizes, which can be difficult to achieve. (Refer to the Broads Authority Web site in Figures 2-8 through 2-11 in Chapter 2 as an example.) Figure 7-15 shows a simple flexible layout at two widely different screen resolutions. Notice how the content adapts to the different resolutions. With a simple layout, flexible Web sites resize gracefully, but with more complex content and page designs, you may want to restrict your design with the min-width and max-width properties described in the next section.



1024 x 768 resolution



1366 x 768 resolution

Figure 7-15 Flexible layout adapts to different screen resolutions

Controlling Flexible Layout Width

You can control the compression and expansion of your content in a flexible layout by setting minimum and maximum widths for your content. As you saw in Chapter 6, the min-width and max-width properties let you set these properties for an element. You can add a wrapper element that contains the page elements, and then set the minimum and maximum sizes to control the boundaries of the page layout.

In the following style rule for the wrapper, the min-width value stops the shrinking of the layout at 750 pixels, which is an optimum width for an 800 x 600 monitor; the max-width value stops the expansion of the layout at 1220 pixels, the optimum width for a 1280 x 1024 display. Any monitor with a higher resolution will display the layout at 1220 pixels wide, maintaining the integrity of the layout.

```
div.wrapper {
   width: 100%;
   min-width: 750px;
   max-width: 1220px;
}
```

You would then apply the wrapper to a element that contains all other elements in your page layout. The following code shows a simplified version of what the wrapper looks like in HTML:

```
<div id="wrapper">  <!–opens wrapper -->

<div id="header">   header content...    </div>
<div id="nav">   nav content...    </div>
<div id="article">   article content...    </div>

</div>  <!–closes wrapper -->
```

You will build a wrapper division to contain a page layout like this in the Hands-on Activities later in the chapter.

To avoid guessing screen measurements, use a shareware tool such as Screen Ruler, available from *www.microfox.com*, to easily measure pixels on your computer monitor.

Activity: Creating a Flexible Layout

In the following steps, you will build a flexible two-column layout to practice using floating elements within a flexible design. As you work through the steps, refer to Figure 7-19 to see the results you will achieve. New code that you will add is shown in **blue**. Save your file and test your work in a browser as you complete each step.

To create the flexible layout:

1. Copy the **flexible_layout.htm** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)
2. Open the file **flexible_layout.htm** in your HTML editor, and save it in your work folder as **flexible_layout1.htm**.
3. In your browser, open the file **flexible_layout1.htm**. When you open the file, it looks like Figure 7-16.

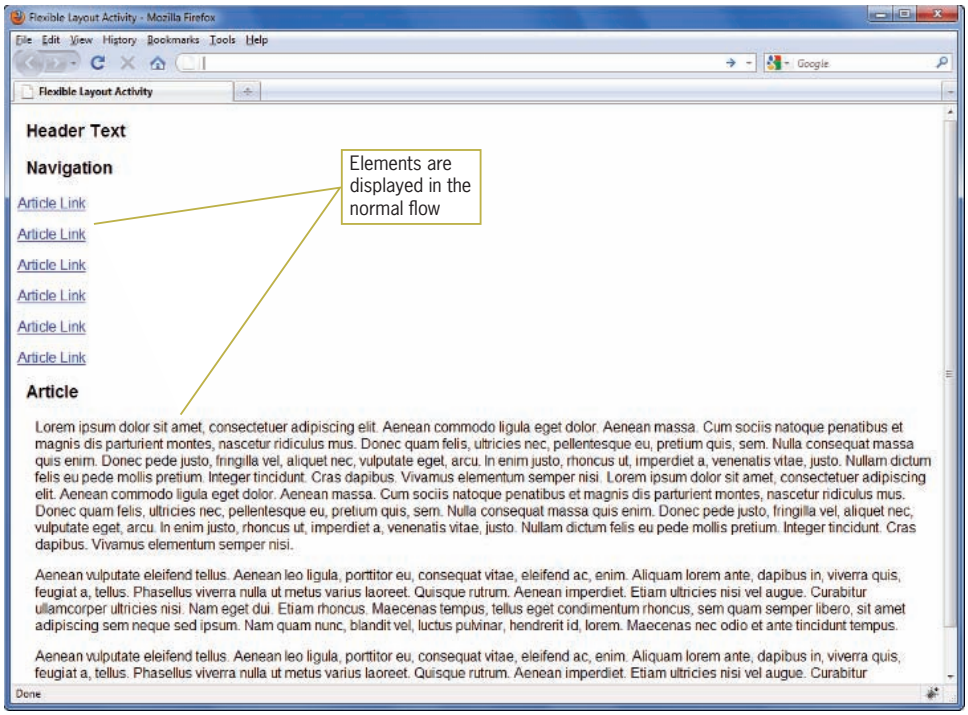


Figure 7-16 Flexible layout activity Web page

When you first look at the style rule you only see the basic formatting properties as shown:

```
<style type="text/css">
body {font-family: arial;}
h3 {font-family: arial; margin-left: 10px;}
.copy {margin-left: 20px}
</style>
```

There are no style rules for any of the content elements, so all are displayed in the normal flow, top to bottom in order and left-aligned to the browser window.

Each division element in the code already has the id names in place, as shown in the code for the header section:

```
<div id="header"><h3>Header Text</h3></div>
```

Formatting the Header

1. Begin by formatting the header division. Within the <style> element, add a style rule that selects the id named *header* and sets the height to 80 pixels.

```
#header {
height: 80px;
}
```

2. Add margins to offset the header from the top of the browser window and the content below by 10 pixels.

```
#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
}
```

3. Align the text to the center of the header, add a solid thin black border, and apply light purple as the background color (#e5dfec).

```
#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
   text-align: center;
   border: solid thin black;
   background-color: #e5dfec;
}
```

4. Save your file and preview the work in the browser. The header is now formatted, as shown in Figure 7-17.

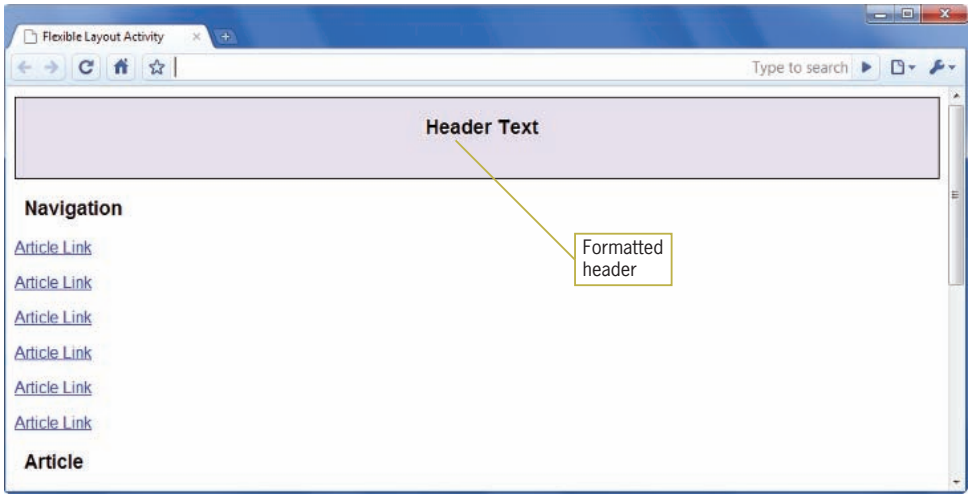


Figure 7-17 Completed header division

Formatting the Nav Division

1. Add a new selector for the id named *nav*, and set the division to float left with a width of 15%.

```
#nav {
   float: left;
   width: 15%;
}
```

2. Set the height to 500 pixels and align the text to center.

```
#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
}
```

3. Add a solid thin black border and set the background color to light orange (#fabf8f).

```
#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
   border: solid thin black;
   background-color: #fabf8f;
}
```

4. Save your file and preview the work in the browser. The nav division is now formatted, as shown in Figure 7-18. The article division needs some work, which you will do in the next set of steps.

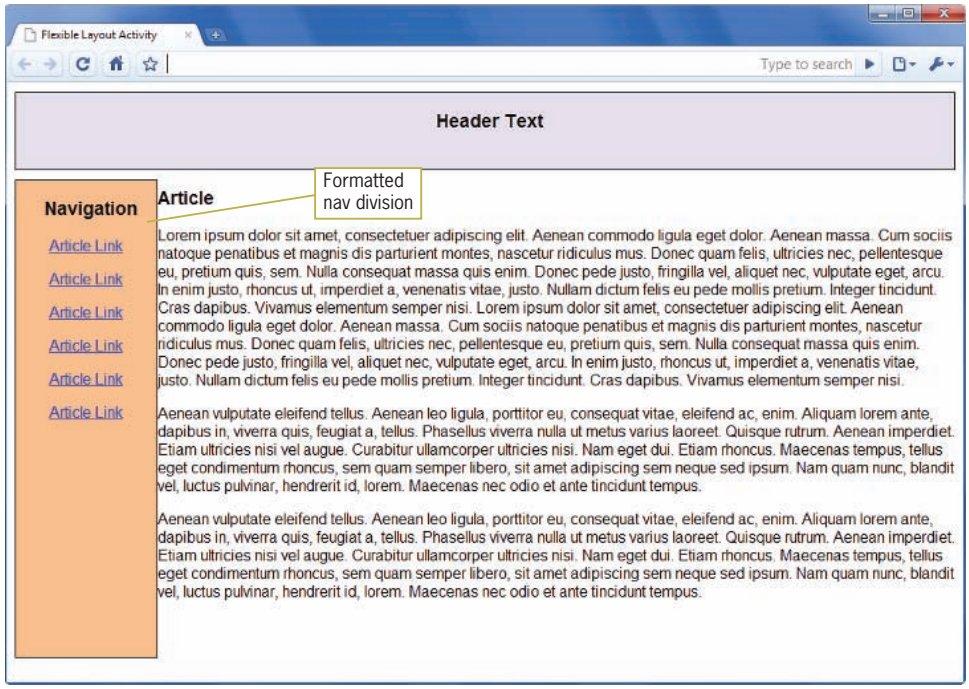


Figure 7-18 Completed nav division

Formatting the Article Division

1. Within the <style> element, add a style rule that selects the id named *article* and sets it to float left with a width of 80%.

```
#article {
   float: left;
   width: 80%;
}
```

2. Set a left margin of 2%. This allows the margin to adapt to different browser widths. Also set a background-color of white (#fff).

```
#article {
   float: left;
   width: 80%;
   margin-left: 2%;
   background-color: #fff;
}
```

3. Save your file and preview the work in the browser. The article division is now formatted, as shown in Figure 7-19, which completes the Web page layout. Notice as you shrink and expand the browser window the content adapts to fill the window.

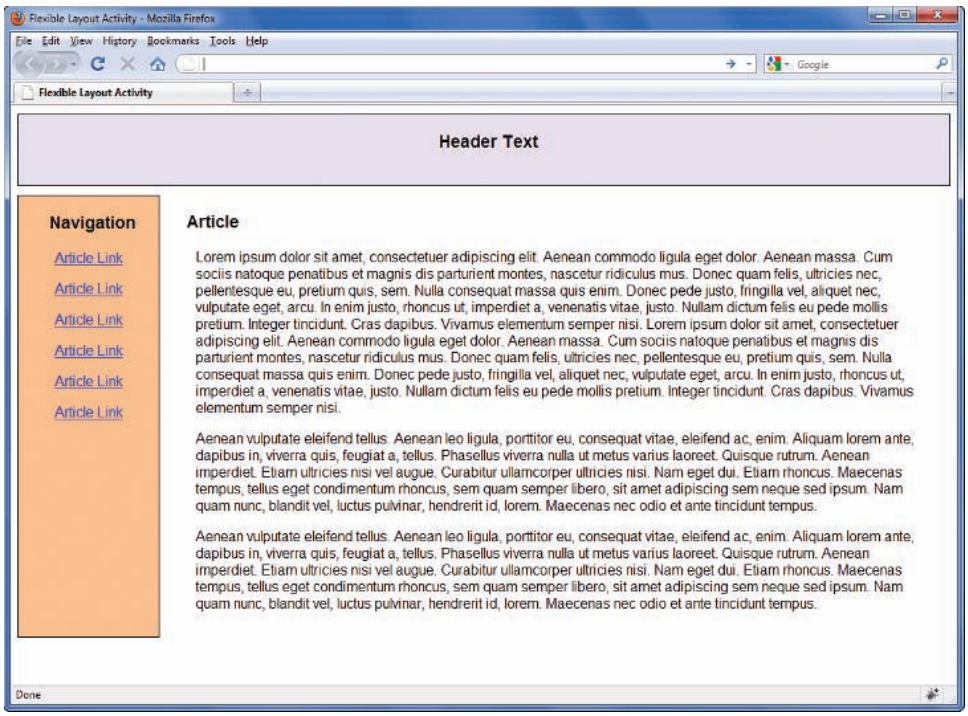


Figure 7-19 Completed flexible layout

The complete style sheet for the Web page in Figure 7-19 is as follows:

```
<style type="text/css">
body {font-family: arial;}
h3 {font-family: arial; margin-left: 10px;}
.copy {margin-left: 20px}

#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
   text-align: center;
   border: solid thin black;
   background-color: #e5dfec;
}

#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
   border: solid thin black;
   background-color: #fabf8f;
}

#article {
   float: left;
```

```
    width: 80%;
    margin-left: 2%;
    background-color: #fff;
    }
</style>
```


Building a Fixed Page Layout

Fixed layouts remain constant despite the resizing of the browser in different screen resolutions and monitor sizes. Many designers prefer fixed layouts because they have more control over the finished design. They can also build more complex layouts because they can be fairly sure of consistent results. Fixed layouts are normally contained by a wrapper element that controls the page width and centers the page in the browser window. Within the wrapper, you can choose whether to contain only fixed-size elements, percentage elements, or a combination of the two. Because the outside width of the page is fixed, the design is more precise, and content can flow down the page as necessary. Pixel measurements are favored by many designers when creating fixed designs.

Refer to Figures 2-12 and 2-13 (The Boston Vegetarian Society Web site) for examples of a fixed layout.

Figure 7-20 shows a two-column layout that contains four elements.

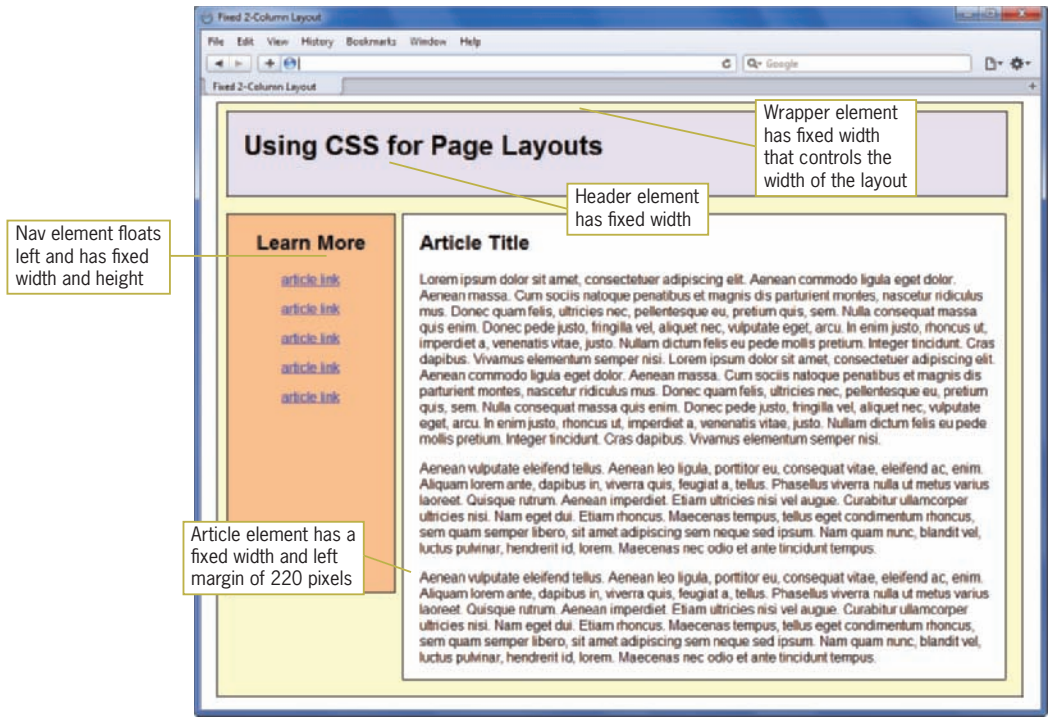


Figure 7-20 Two-column fixed layout

A wrapper division contains the other content elements and sets the fixed width for the layout. Header, nav, and article elements contain the page content. The nav element floats to the left and has a fixed width and height. The article element has a margin-left property that positions it on the page.

The style rules for these four elements follows:

```
#wrapper {
   width: 960px;
   margin-right: auto;
   margin-left: auto;
   border: thin solid black;
   background-color: #ffc;
}

#header {
   width: 930px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
   border: thin solid black;
   background-color: #e5dfec;
}

#nav {
   width: 200px;
   height: 450px;
   float: left;
   border: thin solid black;
   margin: 20px 20px 0px 10px;
   text-align: center;
   background-color: #fabf8f;
}

#article {
   width: 718px;
   border: thin solid black;
   margin: 20px 8px 20px 220px;
   background-color: #fff;
}
```

Notice that the wrapper element has a fixed width, but no fixed height, allowing content to flow down the page as necessary. The 960 pixel value for the width reflects the current base screen resolution of 1024 x 768; this may change based on your user's needs.

The header is in the normal flow and has a fixed width and height. The nav division floats left and has a fixed width and height as well. This element can be made more flexible by removing the height property and letting the content determine the height of the element.

The article element has a fixed width but no height, allowing the height of the element to be based on the amount of its content. The article element has a left margin (the last value in the margin property) that offsets the article 220 pixels from the left side of the browser window.

Controlling Fixed Layout Centering

Another benefit of using a wrapper division to contain your layout is the ability to automatically center the layout horizontally in the browser. This is a great solution for wide-screen monitors, as your layout will always be centered regardless of the screen resolution. Automatic centering is a simple use of the margin property. In the following style rule for the wrapper division in Figure 7-20, the margin-left and margin-right properties are set to *auto*, telling the browser to automatically proportion the extra space in the browser window, resulting in a centered layout.

```
#wrapper {
   width: 960px;
   margin-left: auto;
   margin-right: auto;
   border: thin solid black;
   background-color: #ffc;
}
```

Figure 7-21 shows the centered two-column fixed layout in a 1366 x 768 screen resolution.

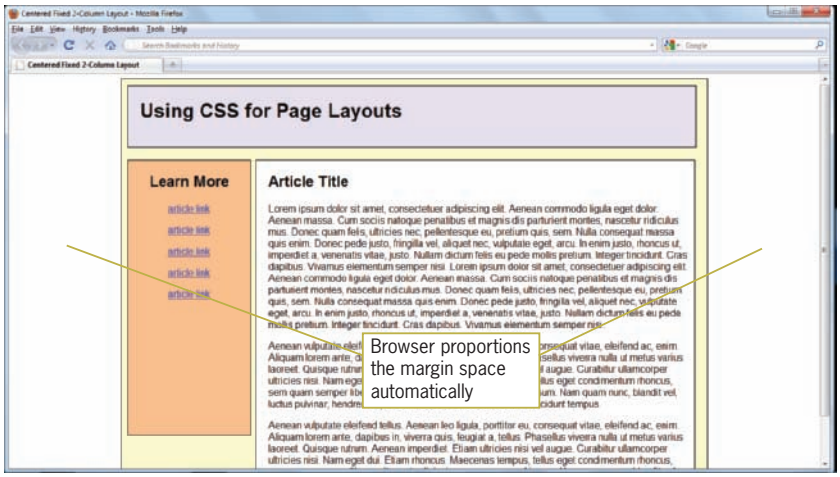


Figure 7-21 Centered two-column fixed layout

Activity: Creating a Fixed Layout

In the following steps, you will build a fixed three-column layout to practice using floating elements within a fixed design. As you work through the steps, refer to Figure 7-28 to see the results you will achieve. New code that you will add is shown in **blue**. Save your file and test your work in a browser as you complete each step.

To create the fixed layout:

1. Copy the **fixed_layout.htm** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)
2. Open the file **fixed_layout.htm** in your HTML editor, and save it in your work folder as **fixed_layout1.htm**.
3. In your browser, open the file **fixed_layout1.htm**. When you open the file it looks like Figure 7-22.

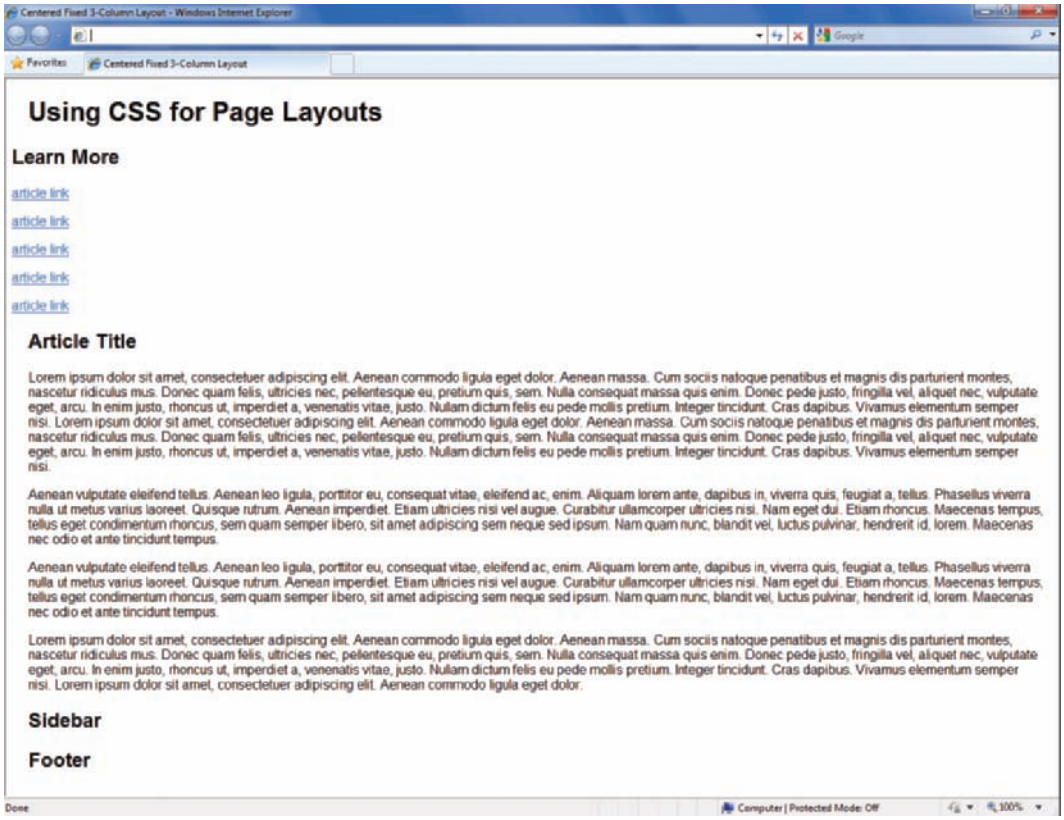


Figure 7-22 Fixed layout activity Web page

When you first look at the CSS style rules, you only see the basic formatting rules as shown:

```
<style type="text/css">
body {font-family: arial;}
h1 {margin-left: 20px;}
.copy {margin-left: 20px; margin-right: 10px;}
.title {margin-left: 20px;}
</style>
```

There are no style rules for any of the content elements, so all are displayed in the normal flow, top to bottom in order and left-aligned to the browser window.

Creating the Wrapper Division

1. Start the project by creating the wrapper division element that will contain the Web page content. Add a style rule that selects an id named *wrapper*. Set the width of the element to 1220 pixels for a 1280 x 1024 resolution.

```
#wrapper {
   width: 1220px;
}
```

2. Add margin-right and margin-left properties, and set these to *auto* so the page is automatically centered in the browser window.

```
#wrapper {
   width: 1220px;
   margin-right: auto;
   margin-left: auto;
}
```

3. Add a thin solid border and a background color of light yellow (#ffc) to complete the wrapper style rule.

```
#wrapper {
   width: 1220px;
   margin-right: auto;
   margin-left: auto;
   border: thin solid;
   background-color: #ffc;
}
```

4. Save your work and view it in the browser. Figure 7-23 shows the completed wrapper division.

If you are working with a monitor that has a lower resolution, such as 1024 x 768, you can adjust the width to a smaller width measurement, say 960 pixels. You will also have to adjust the widths of the contained division elements accordingly.

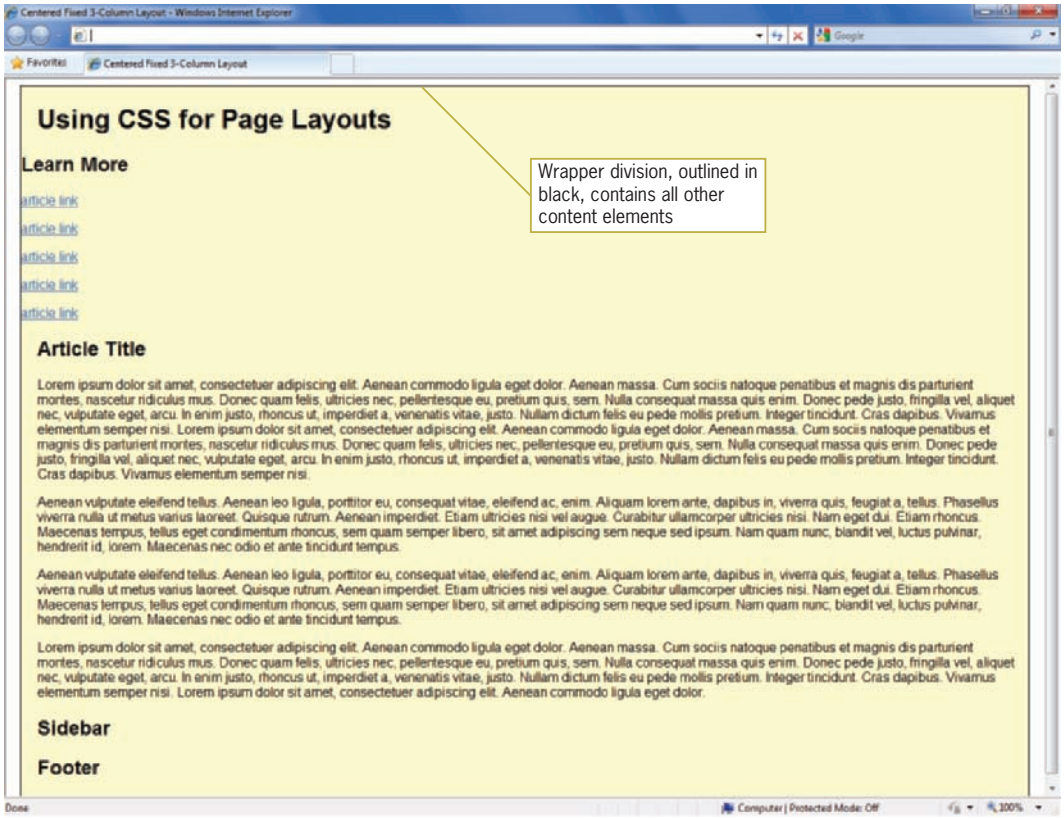


Figure 7-23 Completed wrapper division

Creating the Header Division

1. Add a style rule that selects an id named *header*. Set the width to 1200 pixels and the height to 100 pixels.

```
#header {
   width: 1200px;
   height: 100px;
}
```

2. Add a top margin of 10 pixels and a left margin of 10 pixels.

```
#header {
   width: 1200px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
}
```

3. Finish the header division by adding a solid thin border and a background color of light purple (#e5dfec).

```
#header {
   width: 1200px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
   border: thin solid;
   background-color: #e5dfec;
}
```

4. Save your work and view it in the browser. Figure 7-24 shows the completed header division.

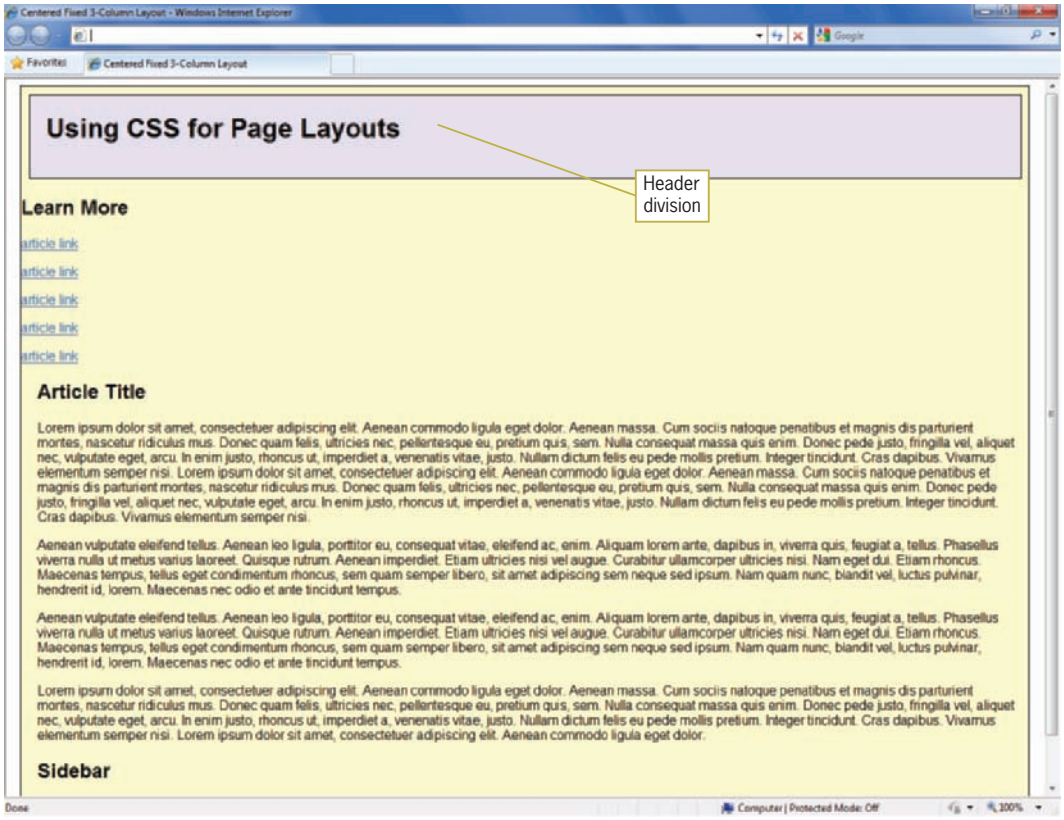


Figure 7-24 Completed header division

Creating the Nav Division

1. Add a style rule that selects an id named *nav*. Set the width to 200 pixels and the height to 600 pixels.

```
#nav {
   width: 200px;
   height: 600px;
}
```

2. Float the nav division to the left. Use the margin shorthand properties to set the following margin values:

   - Top margin: 20 pixels
   - Right margin: 14 pixels
   - Bottom margin: 0 pixels
   - Left margin: 10 pixels

```
#nav {
   width: 200px;
   height: 600px;
   float: left;
   margin: 20px 14px 0px 10px;
}
```

3. Complete the nav division by aligning the text to the center, adding a thin solid border, and setting the background color to light orange (#fabf8f).

```
#nav {
   width: 200px;
   height: 600px;
   float: left;
   margin: 20px 14px 0px 10px;
   text-align: center;
   border: thin solid;
   background-color: #fabf8f;
}
```

4. Save your work and view it in the browser. Figure 7-25 shows the completed nav division.

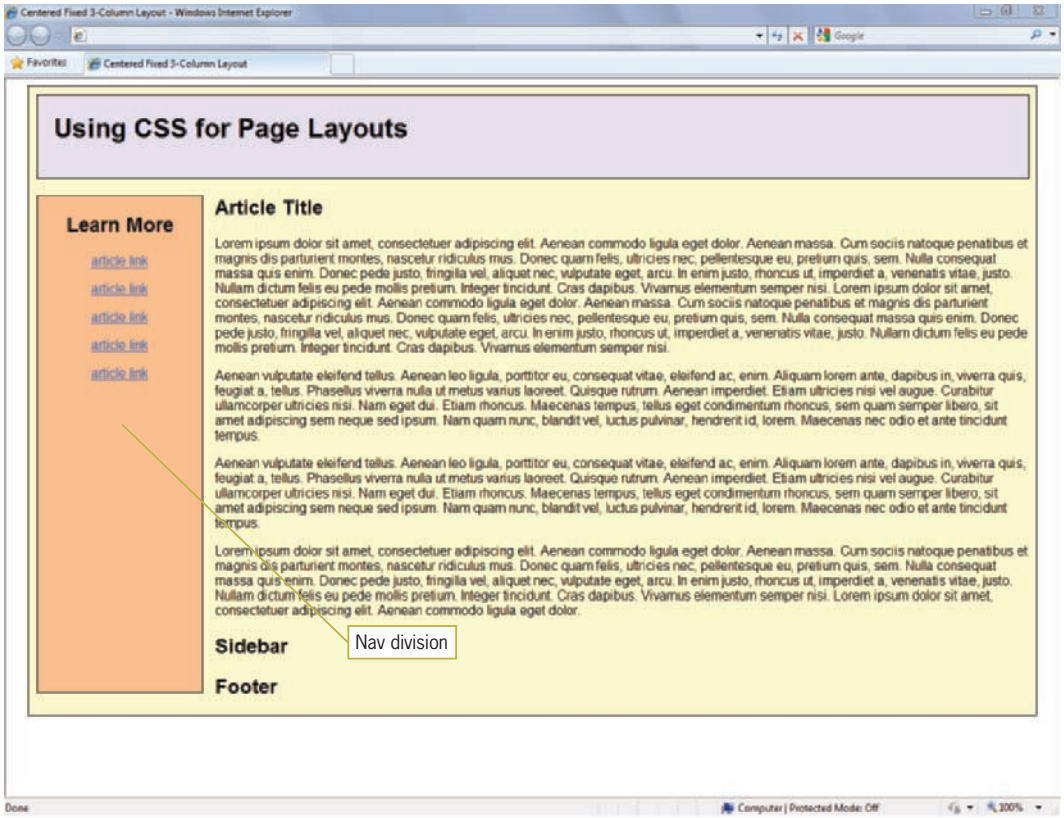


Figure 7-25 Complete nav division

Creating the Article Division

1. Add a style rule that selects an id named *article*. Set the width to 778 pixels and float the division to the left.

```
#article {
   width: 778px;
   float: left;
}
```

2. Use the margin shorthand properties to set the following margin values:
   - Top margin: 20 pixels
   - Right margin: 0 pixels

- Bottom margin: 20 pixels
- Left margin: 0 pixels

```
#article {
   width: 778px;
   float: left;
   margin: 20px 0px 20px 0px;
}
```

3. Add a thin solid border and a background color of white (#fff) to complete the article division.

```
#article {
   width: 778px;
   float: left;
   margin: 20px 0px 20px 0px;
   border: thin solid;
   background-color: #fff;
}
```

4. Save your work and view it in the browser. Figure 7-26 shows the completed article division.

   Notice that the floating elements are no longer properly contained within the wrapper division. You will fix this when you create the footer division at the end of this activity.

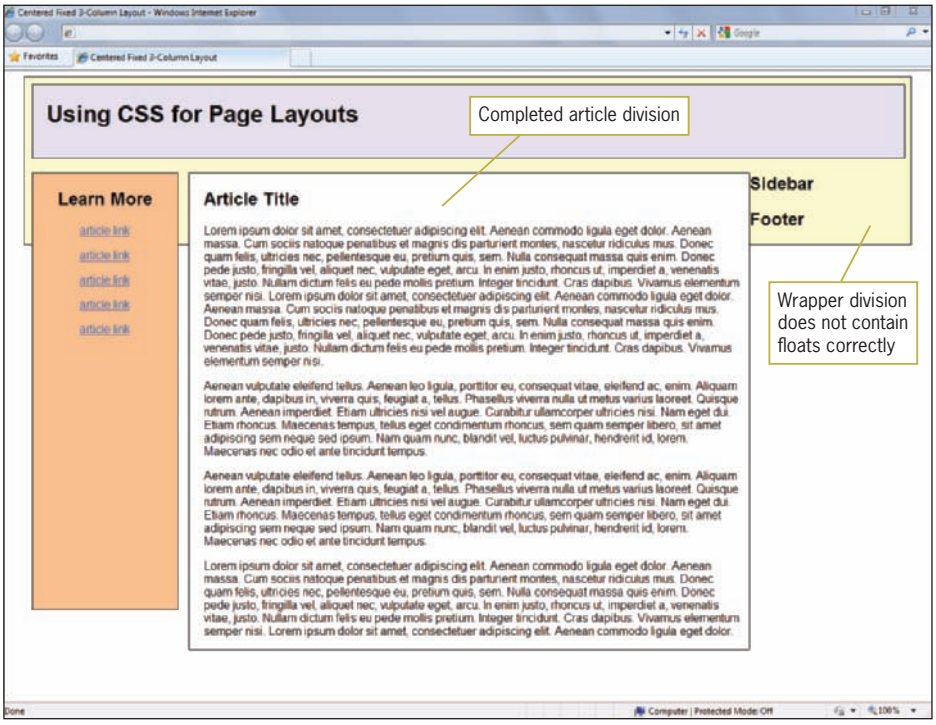


Figure 7-26 Completed article division

Creating the Sidebar Division

1. Add a style rule that selects an id named *sidebar*. Set the width to 200 pixels and the height to 600 pixels.

```
#sidebar {
   width: 200px;
   height: 600px;
}
```

2. Float the sidebar division to the right.

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
}
```

3. Use the margin shorthand properties to set the following margin values:

   - Top margin: 20 pixels
   - Right margin: 8 pixels
   - Bottom margin: 0 pixels
   - Left margin: 0 pixels

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
   margin: 20px 8px 0px 0px;
}
```

4. Add a thin solid border and a background color of light blue (#c6d9f1) to complete the sidebar division.

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
   margin: 20px 8px 0px 0px;
   border: solid thin;
   background-color: #c6d9f1;
}
```

5. Save your work and view it in the browser. Figure 7-27 shows the completed sidebar division. The floating elements are still not properly contained within the wrapper division.

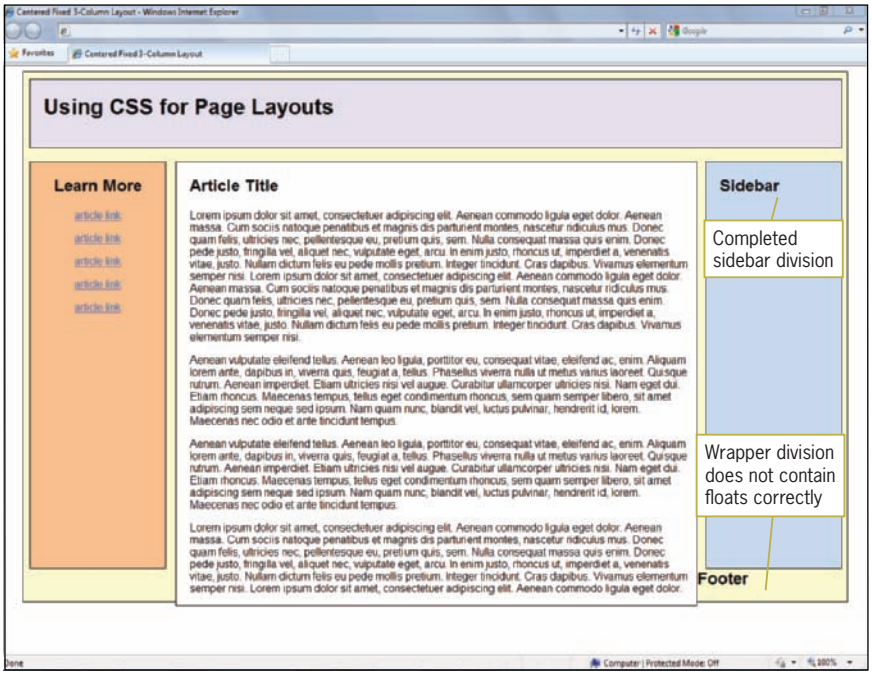


Figure 7-27 Completed sidebar division

Creating the Footer Division and Containing the Floats

1. Add a style rule that selects an id named *footer*. Set the width to 1200 pixels and the height to 75 pixels.

```
#footer {
   width: 1200px;
   height: 75px;
}
```

2. Add the clear property and set the value to *both*. This will fix the float containment problem you have seen in the previous steps.

```
#footer {
   width: 1200px;
   height: 75px;
   clear: both;
}
```

3. Finish the footer by adding a solid thin border, setting the left and bottom margins to 10 pixels, and adding a background color of pink (#efdfec).

```
#footer {
   width: 1200px;
   height: 75px;
   clear: both;
   border: solid thin;
   margin-left: 10px;
   margin-bottom: 10px;
   background-color: #efdfec;
}
```

4. Save your work and test the file in the browser. You will see that the wrapper now contains all of the floating elements, as shown in Figure 7-28.

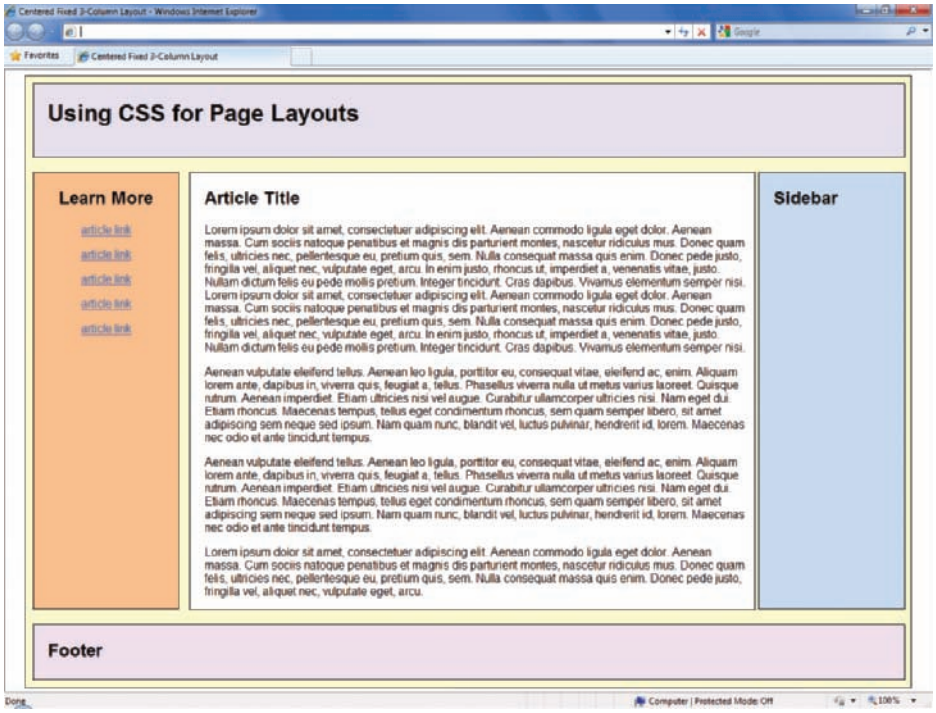


Figure 7-28 Completed fixed layout

Chapter Summary

In this chapter, you learned to apply the CSS box properties to Web page design. You learned about the normal flow of elements in the document and how the float property lets you remove elements from the normal flow. You learned how to create both flexible and fixed layouts, and to resolve problems such as dropped columns and incorrectly positioned floats. Finally, you applied what you learned by building two complete page layouts using the box properties.

- The normal flow dictates the way in which elements normally are displayed in the browser window.
- When you remove an element from the normal flow, you may see unexpected behavior from other elements that are following the normal flow.
- Remember to always use a width property for floating elements; otherwise, the element will extend across the page like elements in the normal flow.
- Remember to avoid using the height property unless you are containing elements such as images that do not change size.
- For fixed layouts, content elements are usually contained with a wrapper element that sets the width for the page layout.

Key Terms

column drop—A layout error that occurs when the total width of the columnar elements in a page layout exceeds the width of their containing element.

fixed layout—A layout that remains constant despite the resizing of the browser in different screen resolutions and monitor sizes.

flexible layout—A layout that shifts as the user resizes the window, wrapping text or adding white space as necessary to adapt to the size of the browser window. Also called liquid layouts.

float—To position an element by taking it out of the normal flow of the Web page layout.

normal flow—The sequence of element display in standard HTML from top to bottom and left to right until all elements that make up the Web page have been displayed.

wrapper—A division element designed to contain an entire Web page.

Review Questions

1. What is the normal flow for block-level elements?
2. Which element is the primary tool for creating sections of content in a Web page?
3. What is the common name for the content container for a Web page?
4. Would you normally use the height property when designing Web pages? Why or why not?
5. What property would you use to add gutters between columns of text?
6. What are the two methods of properly containing floats on a Web page?
7. Can you float elements within floating elements?
8. What causes column drops?
9. What property can be used to help position floating elements properly on a Web page?
10. What are the three possible values of the clear property?
11. What is the benefit of flexible layouts?
12. What is the benefit of a fixed layout?
13. What are the properties and values you would use to control flexible layout width?
14. What are the properties and values you would use to automatically center a fixed layout?

Hands-On Projects

1. Create a fixed three-column Web page with a header and a footer as shown in Figure 7-29. This page is designed for a 1024 x 768 resolution.

   a. Copy the **fixed_columns.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

   b. Open the file **fixed_columns.html** in your HTML editor, and save it in your work folder as **fixed_columns1.html**.

   c. In your browser, open the file **fixed_columns1.html** and examine the page layout.

   d. Your goal is to use the values shown in Figure 7-29 to create a finished page with a fixed three-column layout, a header, and a footer.

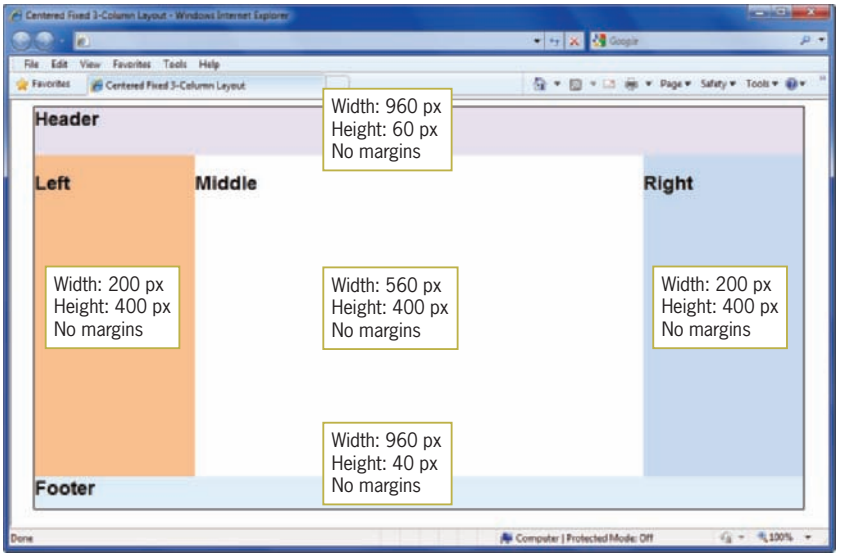


Figure 7-29 Fixed three-column Web page with a header and a footer

2. Fix the column drop shown in Figure 7-30. The completed Web page should look like Figure 7-31.

   a. Copy the **column_drop.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

b. Open the file **column_drop.html** in your HTML editor, and save it in your work folder as **column_drop1.html**.

c. In your browser, open the file **column_drop1.html** and examine the page layout.

d. Your goal is to adjust the values used for the columns to create the Web page shown in Figure 7-31.

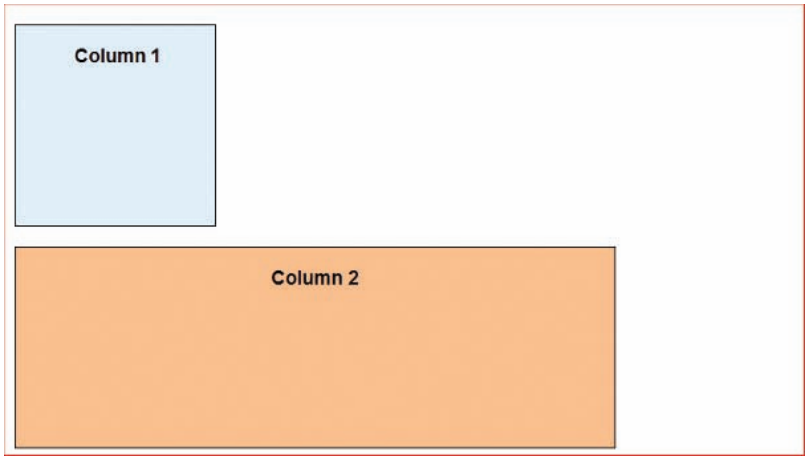


Figure 7-30 Column drop problem

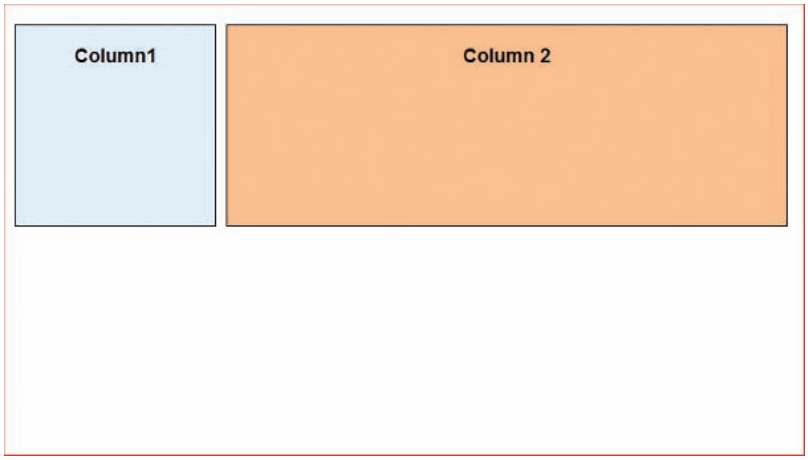


Figure 7-31 Column drop solution

3. Fix the float problem shown in Figure 7-32. The completed Web page should look like Figure 7-33.

   a. Copy the **float_problem.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

   b. Open the file **float_problem.html** in your HTML editor, and save it in your work folder as **float_problem1.html**.

   c. In your browser, open the file **float_problem1.html** and examine the page layout.

   d. Your goal is to use floating and nonfloating elements to create the Web page shown in Figure 7-33.

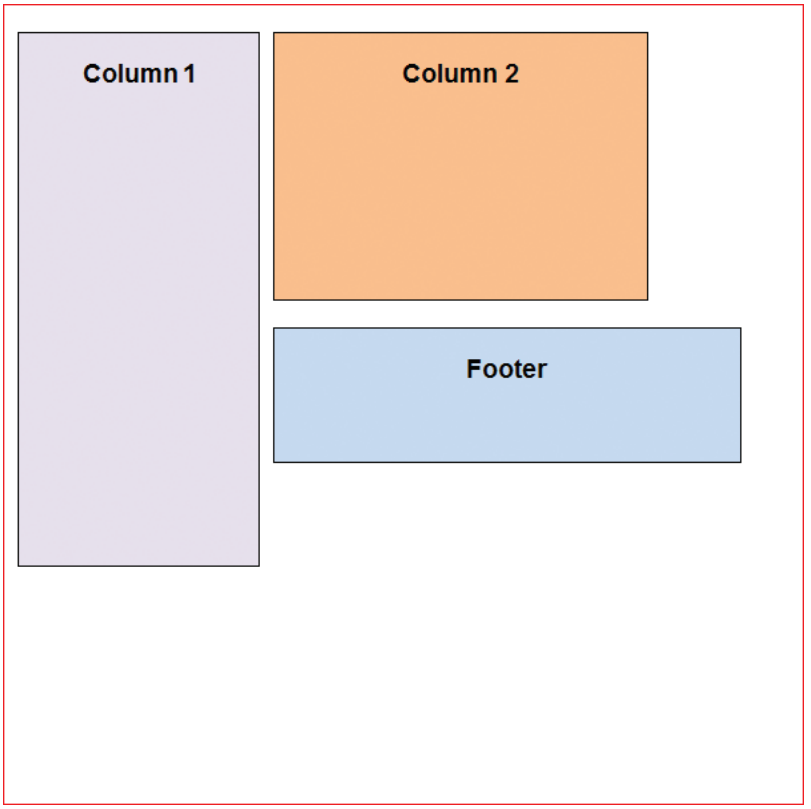


Figure 7-32 Float problem

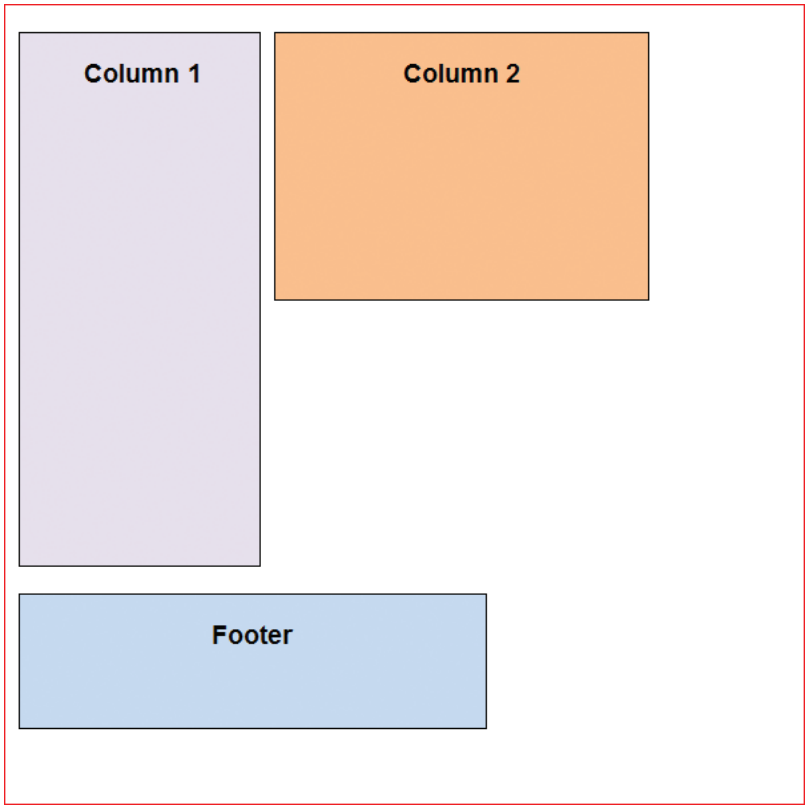


Figure 7-33 Float solution

Individual Case Project

Use your design sketches from Chapter 6 and start to build wireframe page mockups for the different page levels of your site. Decide on whether you are going to build a fixed or flexible design. Using the skills you learned in this chapter, build and submit page layouts for the different levels of information your site will contain. For example, you need to build a home page mockup, an article page mockup, and a section page mockup. Remember to test your page mockups with some text content and at different browser sizes and screen resolutions. Be prepared to explain your choices for your page layouts, such as why you chose fixed or flexible layouts.

Team Case Project

Each team member is responsible for different page templates for the different information levels of your Web site. Using the skills you learned in this chapter, build and submit page layouts for the different levels of information your site will contain. For example, one team member is responsible for the home page design, one for the section page design, and one for the article-level page design. Remember to test your page template using the text content at different browser sizes and screen resolutions. Be prepared to explain your choices for your page layouts, such as why you chose fixed or flexible layouts.

CHAPTER 8

Graphics and Color

When you complete this chapter, you will be able to:

- Understand graphics file formats
- Choose a graphics tool
- Use the <img> element
- Control image properties with CSS
- Understand computer color basics
- Control color properties with CSS
- Control background images with CSS

The ability to freely combine graphics, text, and color into page-type layouts is one feature that makes the Web so attractive and popular, but it also can be the undoing of many Web sites. When you combine these elements wisely, you can produce an attractive and engaging site. Conversely, the use of too many large or complex images, poor color choices, or complicated backgrounds forces users to endure long download times and wade through unreadable text and confusing navigation choices.

Find a good balance between images and text. Use CSS to control image characteristics, such as spacing and text alignment. CSS background images let you enhance page layouts and brand your site.

Use color carefully to communicate, to guide the reader, or to create branded areas of your site. Test your color choices carefully to make sure they appear properly across different browsers. Also, test at a variety of connection speeds to make sure the time needed to download your graphics does not discourage your readers.

Understanding Graphics File Formats

You currently can use three image file formats on the Web: GIF, JPG, and PNG. A fourth format, SVG, has had limited success because it is not supported by Internet Explorer 8 and earlier, although support is promised in Internet Explorer 9. Choosing the right file format for an image is important. If you choose the wrong file type, your image will not compress or appear as you expect.

GIF

The **Graphics Interchange Format (GIF)** is designed for online delivery of graphics. GIF uses a "lossless" compression technique, meaning that no color information is discarded when the image is compressed.

The color depth (described in the "Understanding Computer Color Basics" section of this chapter) of GIF is 8-bit, allowing a palette of no more than 256 colors. The fewer colors you use, the greater the compression and the smaller the file size. The GIF file format excels at compressing and displaying flat (unshaded) color areas, making it the logical choice for line art (simple drawings) and color graphics. Because of its limited color depth, however, GIF is not the best file format for photographs or more complex graphics that have gradations of color, such as shadows and feathering.

GIF Transparency

With GIF files you can choose one color in an image to appear as transparent in the browser. The background color or pattern of the page will show through the areas in the image that you have designated as transparent. Using transparent areas allows you to create graphics that appear to have an irregular outside shape, rather than a rectangular shape. Figure 8-1 shows the same shape with and without transparency.

You can create transparent areas using a graphics editor. When you choose the transparent color, all pixels of that color in the image let the background color show through. In Figure 8-1, the top image has no transparency. In the bottom image, the white background has been made transparent in an image-editing program, and the page color shows through the transparent areas of the graphic.

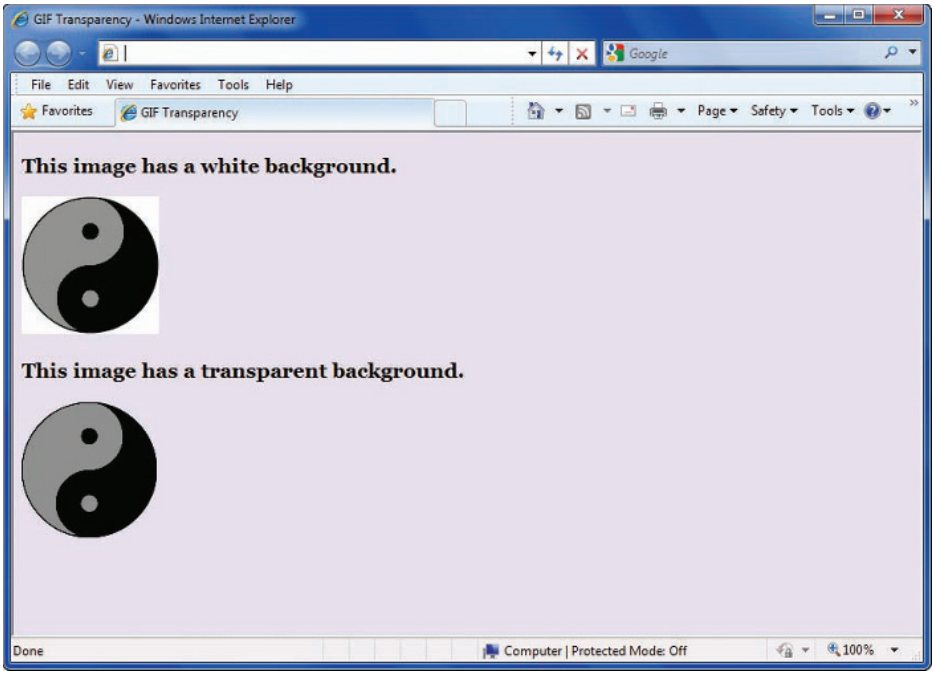


Figure 8-1 Transparent and nontransparent GIFs

GIF Animation

The GIF format lets you store multiple images and timing information about the images in a single file. This means that you can build animations consisting of multiple static images that change

continuously, creating the illusion of motion. You can create animated GIFs by using a variety of shareware and commercial software. Animated GIFs were very popular in the earlier days of the Web, but they have gone out of favor in modern Web design.

When you create a GIF animation, you can determine the time between frames and the number of times the animation plays. Figure 8-2 shows a series of individual GIFs combined to play as one animated GIF. The final GIF animation file is a single file whose name ends in the .gif extension.

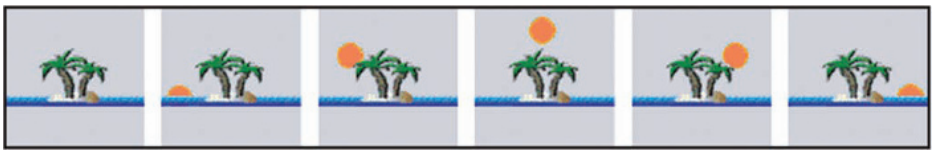


Figure 8-2 Individual frames of a GIF animation

GIF animation is somewhat limited when compared with the results of proprietary animation tools such as Adobe Shockwave or Flash, which can play synchronized sounds and allow Web users to interact with the animation. However, unlike most proprietary tools, animated GIFs do not require any special plug-ins for viewing. Also, if you limit color and motion when creating your animations, you can keep your file sizes small for faster downloads.

Use restraint when adding animated GIFs such as blinking icons and scrolling banners to your pages; users may find them annoying because they are repetitive and distract from the page content. Consider choosing to play an animation a limited number of times rather than letting it loop endlessly. Creating animated images with GIF animation software streamlines the process of setting the timing, color palette, and individual frame effects. See Table 8-1 for a list of GIF animation tools.

GIF Animation Tool	URL
GIF Construction Set Professional	*www.mindworkshop.com/gifcon.html*
GIFMation	*www.boxtopsoft.com/gifmation.html*
Advanced GIF Animator	*www.gif-animator.com*

Table 8-1 GIF Animation Tools

JPG

The **Joint Photographic Experts Group** (**JPG**, sometimes called **JPEG**) format is best for photographs or continuous-tone images. JPGs are 24-bit images that allow millions of colors. Unlike GIFs, JPGs do not use a palette to display color.

JPGs use a "lossy" compression routine specially designed for photographic images: when the image is compressed, some color information is discarded, resulting in a loss of quality from the original image. Because the display device is a low-resolution computer monitor, the loss of quality is not usually noticeable. Furthermore, JPG's faster download time compensates for its loss of image quality.

Using Adobe Photoshop or other imaging software, you can translate photographic images into the JPG format. When you create the JPG file, you can balance the amount of compression versus the resulting image quality manually. Figure 8-3 shows the Photoshop Save for Web & Devices dialog box, which is the tool you use in Adobe Photoshop to adjust quality and find the estimated download time.

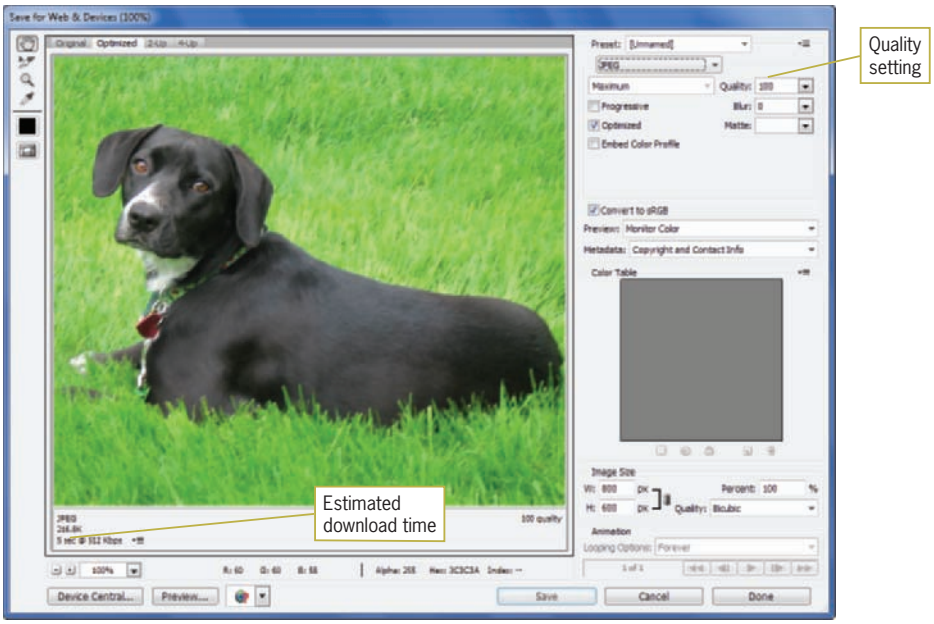


Figure 8-3 Photoshop Save for Web & Devices dialog box

Whether you are creating GIFs or JPGs, always remember to save an original copy of your artwork or photo. Both file formats permanently degrade the quality of an image as a result of compression. Once you have converted to GIF or JPG, you cannot return to the original image quality.

The Quality list box lets you adjust the quality of the file; the higher the quality, the lower the file compression. You can play with this setting to create good-looking files that are as small as possible. Many photos can sustain quite a bit of compression and still maintain image integrity. The Preview window shows the result of your changes, allowing you to experiment with the image quality before saving the file. Photoshop displays the estimated download time based on the file size.

PNG

The **Portable Network Graphics (PNG)** format is designed specifically for the Web. PNG is a royalty-free file format that is intended to replace GIF. This lossless format compresses 8-bit images to smaller file sizes than GIF. PNG supports greater color depths than GIF, so it supports 8-bit indexed color, 16-bit gray scale, and 24-bit true-color images. Even though PNG supports 24-bit color, its lossless compression routine does not compress as efficiently as JPG, so it is not the best choice for photographic images.

PNG supports transparency and interlacing, but not animation. (**Interlacing** is the gradual display of a graphic in a series of passes as the data arrives in the browser.) One useful feature of PNG is its built-in text capabilities for image indexing, allowing you to store a string of identifying text within the file itself.

SVG

The **Scalable Vector Graphics (SVG)** format is a language for describing two-dimensional graphics using XML. SVG files can contain shapes such as lines and curves, images, text, animation, and interactive events. SVG is compatible with common Web technologies such as HTML, XML, JavaScript, and Cascading Style Sheets (CSS). For more information on SVG, visit the W3C's SVG page at *www.w3.org/Graphics/SVG*.

SVG graphics are scalable to different display resolutions and can be printed on high-resolution printers. An SVG graphic can be reused at different sizes throughout a Web site without downloading multiple files to the user. SVG graphics can be viewed at different sizes based on user needs, allowing magnification of an image to see fine detail or to increase legibility.

SVG is a vector graphics file format. **Vector graphics** represent images as geometrical formulas, as compared with the **raster graphics** format, which represents images pixel by pixel for the entire image. GIFs and JPGs are raster formats. The vector graphics format allows SVG graphics to be scalable and cross-platform compatible.

All computer displays, whether desktop or handheld, are raster-type devices. The conversion of vector-based SVG files to pixels is based on the individual display type and settings, resulting in images that reproduce more faithfully for the greatest number of users.

All major modern Web browsers except Microsoft Internet Explorer (IE) support and render SVG graphics. The next major version of IE, Internet Explorer 9, promises support for the SVG format.

Using Interlacing and Progressive Display

Most Web-capable graphics editors let you save images in an interlaced (progressive) format. You can choose this display option when creating GIF, PNG, and JPG files. GIF and PNG files use an interlacing format, while JPG files use a progressive format. Interlacing and progressive formats generally are the same thing—the gradual display of a graphic in a series of passes as the data arrives in the browser. Each additional pass of data creates a clearer view of the image until the complete image is displayed. Figure 8-4 shows three rendering passes to display a complete image.

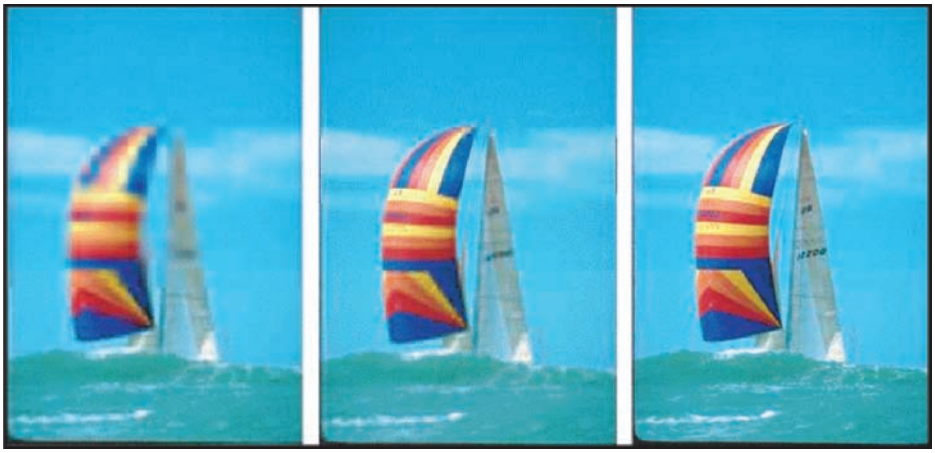


Figure 8-4 Three passes complete this progressive JPG image

The only real advantage to displaying graphics in the interlaced or progressive method is that users immediately see at least a blurred view of the complete image, giving them something to look at while waiting for the entire graphic to download. The disadvantage of choosing this display method is that older browsers may not display the graphic properly, and more processing power is needed on the user's machine to render the image. The use of these methods has declined as increased connection speeds have become more widespread.

Where You Can Find Images

You can acquire images from a variety of sources, including from a graphics professional you hire to create and prepare your images. If your budget does not allow for funding this service, consider one of the following resources:

- *Stock photo collections*—Stock photo collections can cost anywhere from thousands of dollars for a few images to under $20 for thousands of images at your local computer discount store or Web site retailer. These collections contain royalty-free images that you can use for any Web site. You can manipulate the graphics to add or delete text or images, change the color, or make any other modifications. Most stock photo collections include a built-in browsing program that lets you search for a particular image, and some also provide image-editing software.
- *Digital camera*—A digital camera lets you take your own photos and use them on the Web. These cameras store photos in JPG format, so you do not have to convert them. Most also provide image-cataloging software, and some include basic image-editing software. The price of digital cameras continues to drop, while the quality of the images gets better and better.
- *Scanner*—Good scanners are available for under $100. You can scan your own photos or images and save them as GIF, JPG, or PNG files for use on your Web site.
- *Public domain Web sites*—Many Web sites maintain online catalogs of images that are available for download. Some of these sites charge a small membership fee, so you can download as many images as you want. Other public domain Web sites are completely free.

Wikipedia maintains a list of public domain stock photo Web sites at *http://en.wikipedia.org/wiki/Wikipedia:Public_domain_image_resources*

- *Create your own*—If you need a basic image or if you have graphic design skills, you can download a shareware or freeware graphics tool and learn to use it. Keep your custom image simple, such as text on colored backgrounds, and use fundamental shapes and lines. Look at graphics on other Web sites; many are simple but effective and may provide a useful model for your own images.

Do not borrow images from other Web sites. Although your browser allows you to copy graphics, you should never use someone else's work unless it is from a public domain Web site and freely available for use. Digital watermarking technology lets artists copyright their work with an invisible signature; if you use someone else's graphics, you may find yourself in a lawsuit.

Choosing the Right Format

The following list summarizes the advantages and disadvantages of each graphic file format for the Web.

- *GIF*—Still the most common format for all types of simple colored graphics and line art. GIF's transparency feature lets you seamlessly integrate graphics into your Web site.
- *JPG*—Use JPG for all 24-bit full-color photographic images, as well as more complicated graphics that contain color gradients, shadows, and feathering.
- *PNG*—You can use PNG as a substitute for GIF. PNG offers greater compression and color depth than GIF. Because PNG does not compress your 24-bit images as well as JPG does, do not use it for photos.
- *SVG*—Offers many advantages, but lack of support in all major browsers means SVG is not a common image format.

Choosing a Graphics Tool

As a Web designer, you may be in the enviable position of having a complete staff of graphic design professionals preparing graphics for your site. Most Web designers, however, do not have this luxury. Whether you want to or not, you eventually must use a graphics tool. Most of your graphics tasks are simple, such as resizing an image or converting an image from one file format to another. More complex tasks often include changing color depth or adding transparency to an image. These are tasks that anyone can learn using any of the popular graphics software currently available.

When it comes to creating images, you may want to enlist professional help. Your Web site will not benefit if you choose to create your own graphics and you are not up to the task.

Professional-quality graphics can greatly enhance the look of your Web site. Take an honest look at your skills and remember that the best Web sites usually are the result of collaboration.

You use graphics software to create or manipulate graphics. Most Web designers use Adobe Photoshop, which is an expensive and full-featured product that takes time to master. Adobe Illustrator, a high-end drawing and painting tool, also is available. Other commercial tools you can consider include Ulead PhotoImpact and Adobe Fireworks. Most are available as downloadable demos, so you can try before you buy. In general, look for a tool that meets your needs and will not take a long time to learn. Table 8-2 shows a list of Web sites for the graphics tools mentioned in the text.

Graphics Tool	URL
Adobe Photoshop and Illustrator	*www.adobe.com*
Adobe Fireworks	*www.adobe.com*
Corel Paint Shop Pro	*www.corel.com*
Ulead PhotoImpact	*www.ulead.com*

Table 8-2 Graphics Tools Web Sites

The list in Table 8-2 is not exhaustive, and you may have to try different tools to find the one that suits your needs.

Of course, you also can choose from a variety of shareware or freeware graphics tools. One of the more established tools is Paint Shop Pro. This tool is reasonably priced and contains a full range of image-editing features. Like most other shareware, this tool can be downloaded and used for a trial period.

For a list of freeware graphic-editing programs, see: *www.freewarefiles.com/category/graphics.php*

For a list of shareware graphic-editing programs, see: *www.tucows.com/Windows/DesignTools/Image/ImageEditors*

Using the Image Element

By definition, the image element <img> is a replaced element in HTML, meaning that the browser replaces the <img> element with the image file referenced in the src attribute. The browser treats the image as it treats a character; normal image alignment

is to the baseline of the text. Images that are within a line of text must have spaces on both sides or the text will touch the image.

The <img> element only needs the src attribute for the image to be displayed in the browsers, though using only the src attribute is not good coding practice. The <img> tag should always contain additional attributes shown in the following code sample and described in Table 8-3.

```
<img src="logo.gif" width="258" height="130" alt="Company logo
graphic" title="Click the logo to view the company home page" />
```

Attribute	Use
alt	Displays an alternate string of text instead of an image if the user has a text-only browser or has graphics turned off
height	Specifies the height of the image in pixels
src	The only required attribute, src specifies the URL of the graphic file you want to display; as with any URL, the path must be relative to the HTML file
title	A string of text that provides information about the image; visual browsers display the contents of the title attribute as a ToolTip or ScreenTip (a pop-up window that appears when the user pauses the pointing device over an object); an audio browser could speak the title information
width	Specifies the width of the image in pixels

Table 8-3 <img> Element Attributes

Replacing Image Element Attributes with Style Sheet Properties

Much of the HTML code on the Web does not match current standards. When you visit different Web sites and view their code, you often see a variety of older HTML attributes in use to control image characteristics. Specifically, the align, border, vspace, and hspace attributes have been deprecated in HTML 4.01 in favor of CSS. Table 8-4 shows the equivalent CSS properties that replace these attributes.

Deprecated Image Attribute	Equivalent CSS Property
align	The float property allows you to flow text around an image or other object; for example, img {float: left;}
border	The border property lets you set a border on an image or remove the border from a linked image
vspace and hspace	The padding or margin properties set white space around an image; you can control individual sides of the image, or apply white space around the entire image

Table 8-4 CSS Properties that Replace img Attributes

Specifying alt and title Attribute Text

The alt attribute provides a description of the image if the image does not appear in the browser. Proper use of the alt attribute improves Web accessibility by describing the function of each image in your Web site. This information can be used by screen readers and other adaptive devices. Your page layouts should still be readable and navigable even with images turned off. Figure 8-5 shows an example from the Barnes and Noble Web site (*www.bn.com*). Notice that all images have appropriate alt attribute values that let users navigate and understand the site content.

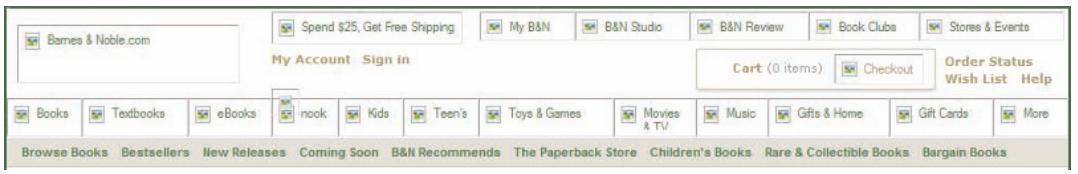


Figure 8-5 Alt text provides navigation and content information

The title attribute contains information about the element like you would see in a ToolTip or pop-up help window. This is usually a description, such as identifying the target of a link, providing copyright or identifying information about an image, or other comments or notations. The following code shows an example of the title attribute used with an <img> element.

```
<img src="balloons_sm.jpg" width="200" height="267" alt="Hot
Air Balloon image" title="Balloons at the Great Falls Festival
in Lewiston, Maine"/>
```

Figure 8-6 shows the pop-up text that appears as a result of using the title attribute.

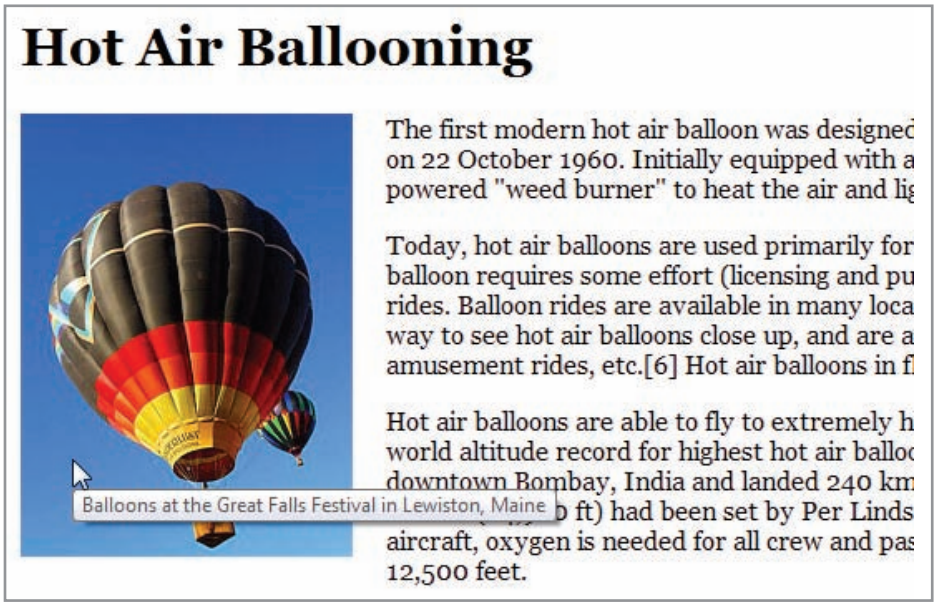


Figure 8-6 Using the title attribute

Specifying Image Width and Height

Every <img> element on your Web site should contain width and height attributes. These attributes provide important information to the browser by specifying the amount of space to reserve for the image. This information dramatically affects the way your pages download, especially at slower connection speeds. If you have included the width and height, the browser knows how much space the image needs. The browser reserves the space on the page without waiting for the image to download, and displays the rest of your text content. If the browser does not know the width and height values, it must download the image before displaying the rest of the page. At slower connection speeds, the user will be looking at a blank page while waiting for the image to download.

You should set the width and height to preserve the look of your layout, whether the images are displayed or not. In Figure 8-7, the width and height have been omitted. Notice that if the browser does not know the width and height, the text wrapping and appearance of the page change dramatically when the image is not displayed

Figure 8-7 Browser unable to reserve image size

The following code shows the width and height attributes for the image. It indicates that the browser should reserve a 200 × 267-pixel space for the balloons_sm.jpg image and should display the alternate text "Hot Air Balloon image" if it cannot display the image.

```
<img src="balloons_sm.jpg" width="200" height="267" alt="Hot
Air Balloon image" title="Up, up and away in a hot air
balloon" >
```

In Figure 8-8, the width and height have been specified and the image size is reserved by the browser, retaining the look of the page layout.

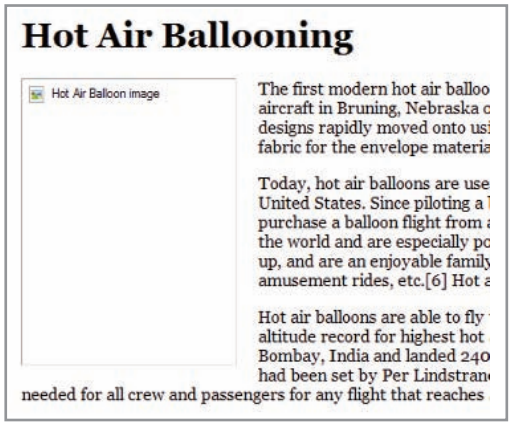


Figure 8-8 Image size reserved in the browser

You may notice that you can manipulate the width and height of the image itself using the width and height attributes in the <img> element. While it is tempting to use these attributes to change a graphic's size without using a graphics program, it is not a good idea. If the original graphic's area is too large and you reduce the size using the width and height attributes, you are not changing the file size of the image—only the area that the browser reserves for the graphic. The user is still downloading the original graphic file; no time is saved. Also, if you do not maintain the ratio of width to height, called the **aspect ratio**, you distort the image. Figure 8-9 shows an image in its actual size, the size after changing the width and height values in proportion to one another, and the distortion caused by incorrect width and height values.

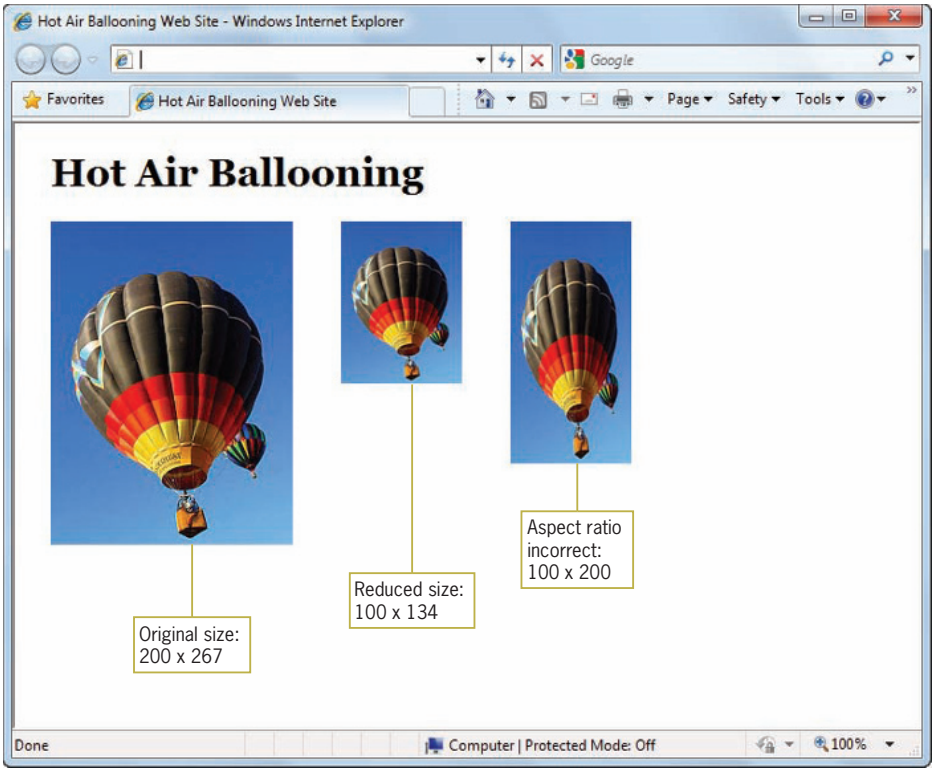


Figure 8-9 Manipulating images with width and height attributes

In the following code for the three images, the width and height attributes appear in bold, colored text:

```
<-- Original size -->
<img src="balloons_sm.jpg" alt="Hot Air Balloon image"
title="Balloons at the Great Falls Festival in Lewiston,
Maine" width="200" height="267" />
```

```
<-- Reduced size -->
<img src="balloons_sm.jpg" alt="Hot Air Balloon image"
title="Balloons at the Great Falls Festival in Lewiston,
Maine" width="100" height="134" />

<-- Incorrect Aspect Ratio -->
<img src="balloons_sm.jpg" alt="Hot Air Balloon image"
title="Balloons at the Great Falls Festival in Lewiston,
Maine" width="100" height="200" />
```

However, the ability to manipulate image size using the width and height attributes comes in handy in certain circumstances. When creating a layout mock-up, you can test image sizes by manipulating the code.

Sizing Graphics for the Page

One way to keep file sizes small is to size graphics appropriately. Few experiences are more annoying than opening a Web page you haven't visited before and waiting to download an overly large image. One of the easiest ways to make your graphics download quickly is to keep their dimensions small and appropriate to the size of the page. Figure 8-10 shows a variety of image sizes at 1024 × 768 screen resolution.

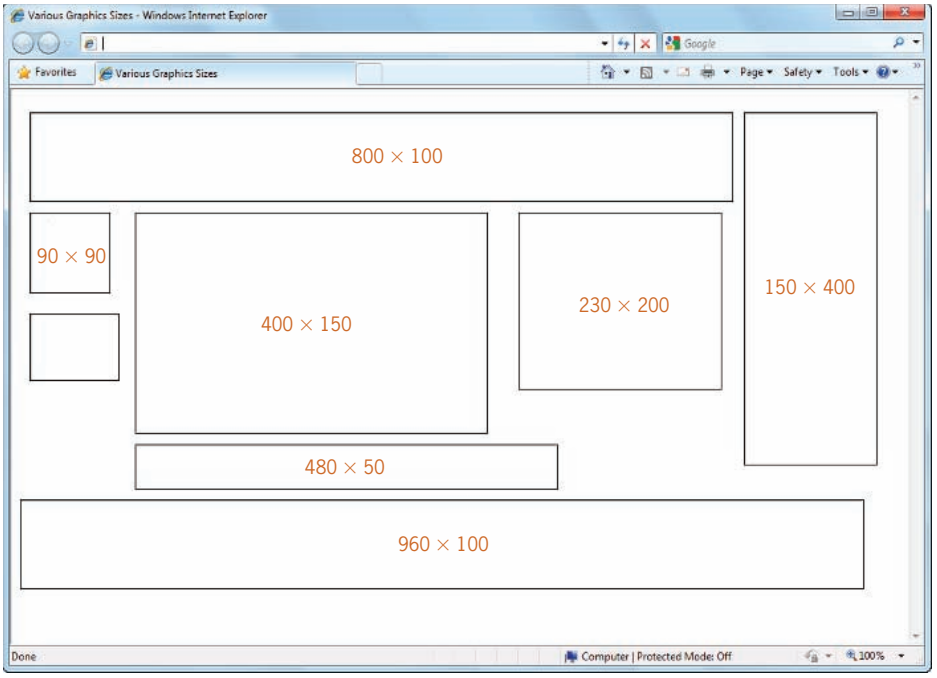


Figure 8-10 Sample image sizes at 1024 × 768 screen resolution

Use these sample image sizes as guidelines when you size your graphics. It is also useful to think of image size in relation to the number of columns in your layout; size your graphics to occupy one, two, or more columns of the page.

Controlling Image Properties with CSS

In this section, you will use Cascading Style Sheet properties to control the following image characteristics:

- Removing the hypertext border
- Aligning text and images
- Floating images
- Adding white space around images

Removing the Hypertext Border from an Image

When you create a hypertext image, the browser's default behavior is to display the hypertext border around the image, as shown in Figure 8-11. This border appears blue before—and purple after—you click the image. In a well-designed site, this border is unnecessary because users often use their mouse to point to each image to see whether the hypertext pointer appears. Another reason to abandon the display of hypertext borders is that their color may not complement your graphic.

To remove the hypertext border, add a style attribute with the border property set to *none*. Here is the code for the second balloon in Figure 8-11, which has the hypertext border turned off:

```
<img src="balloon.jpg" width="100" height="100" alt="balloon"
style="border: none" />
```

You can read more about the border property in Chapter 6.

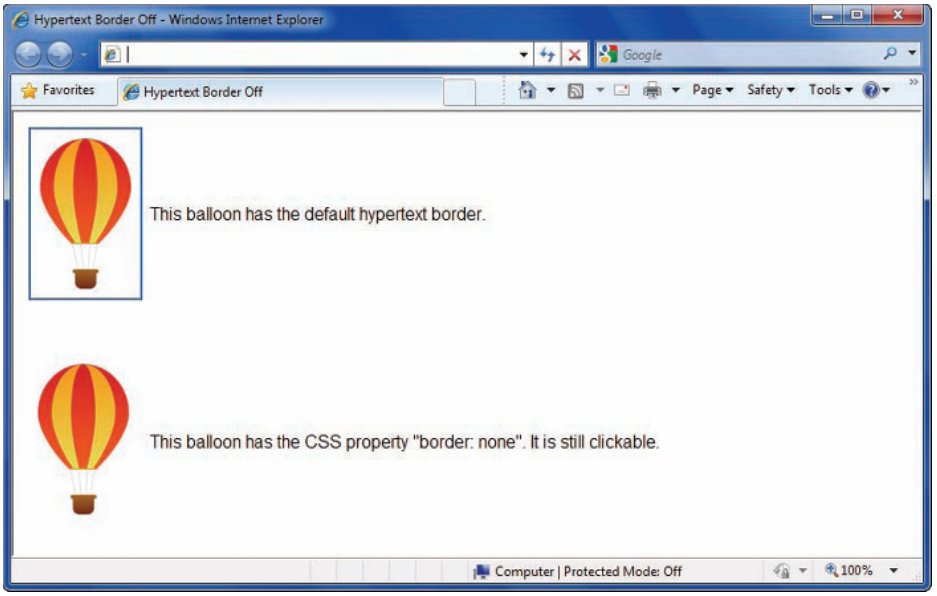


Figure 8-11 Removing the hypertext border from an image

Aligning Text and Images

You can align text along an image border using the vertical-align property. The default alignment of the text and image is bottom-aligned, which means the bottom of the text aligns with the bottom edge of the image. You can change the alignment by using either the top or middle values. Figure 8-12 shows all three alignment values.

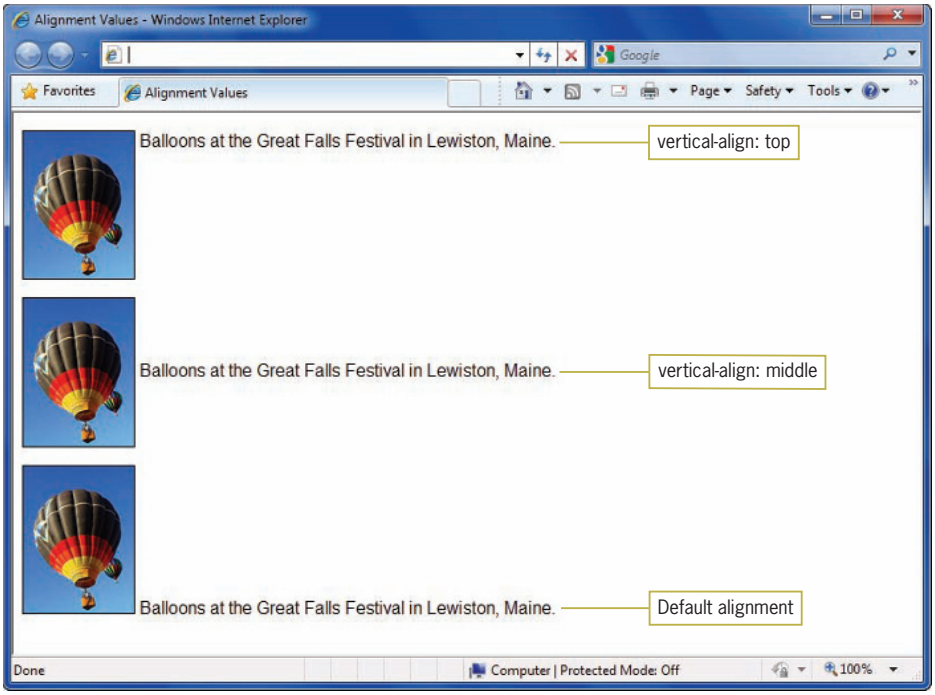


Figure 8-12 Text alignment

Floating Images

float property description	
Value:	left | right | none
Initial:	none
Applies to:	all elements except positioned elements
Inherited:	no
Percentages:	N/A

The float property can be used to float an image to the left or right of text.

The following style rules create two classes of <img> elements, one of which floats to the left of text; the other floats to the right:

```
img.left {float: left;}
img.right {float: right;}
```

You can apply these rules to an image using the class attribute within the <img> element, as shown in the following code fragment:

```
<img src="sample.gif" class="left" />
```

Figure 8-13 shows two floating images within a page.

Figure 8-13 Floating images

Adding White Space Around Images

Add white space around your images to reduce clutter and improve readability. As shown in Figure 8-14, the default spacing is very close to the image.

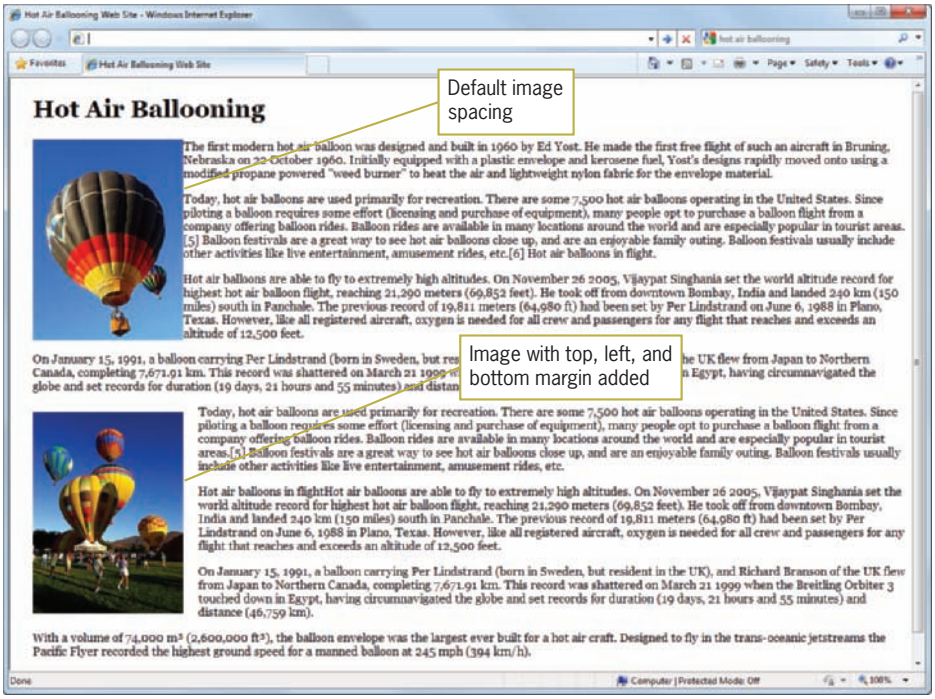


Figure 8-14 Image spacing

Use the CSS margin property to increase the white space around an image. You can read more about the margin property in Chapter 6. The margin property lets you add margins on all four sides or to individual sides of an image. The following code shows an image with a 20-pixel margin on the left, top, and bottom sides, floating to the left of text:

```
img.left {
   float: left;
   margin-left: 20px;
   margin-top: 20px;
   margin-bottom: 20px
}
```

You also can add white space into the graphic itself using graphic-editing software.

Understanding Computer Color Basics

Before you create or gather graphics for your Web site, you need a basic understanding of how color works on computer monitors.

Your computer monitor displays color by mixing the three basic colors of light: red, green, and blue, often called RGB colors. Each of these three basic colors is called a color channel. Your monitor can express a range of intensity for each color channel, from 0 (absence of color) to 255 (full intensity of color). Colors vary widely among monitors based on both the user's preferences and brand of equipment.

Color Depth

The amount of data used to create color on a display is called the **color depth**. If your monitor can display 8 bits of data in each of the three color channels, it has a 24-bit color depth (8 × 3 = 24). 24-bit images can contain almost 17 million different colors and are called true-color images. Both JPG and PNG support 24-bit color. If your users have a 24-bit color display, they can appreciate the full color depth of your images. But many older monitors cannot display 24-bit images; some have only 16-bit color depth (called high color), and some have only 8-bit color depth. If a monitor does not support the full color depth of an image, the browser must resort to mixing colors in an attempt to match the original colors in the image.

Dithering

The browser must mix its own colors when you display a 24-bit image on an 8-bit monitor, or when you use a file format that does not support 24-bit color. Because the 8-bit monitor has fewer colors to work with (256, to be exact), the browser must try to approximate the missing colors by creating colors from the ones the browser already has. This type of color mixing is called dithering. **Dithering** occurs when the browser encounters a color that it does not support, such as when you try to turn a 24-bit photographic image into an 8-bit, 256-color image. Dithered images often appear grainy and pixilated. The dithering is most apparent in gradations, feathered edges, or shadows. Figure 8-15 shows the same image in both JPG and GIF format at 8-bit, 256 colors.

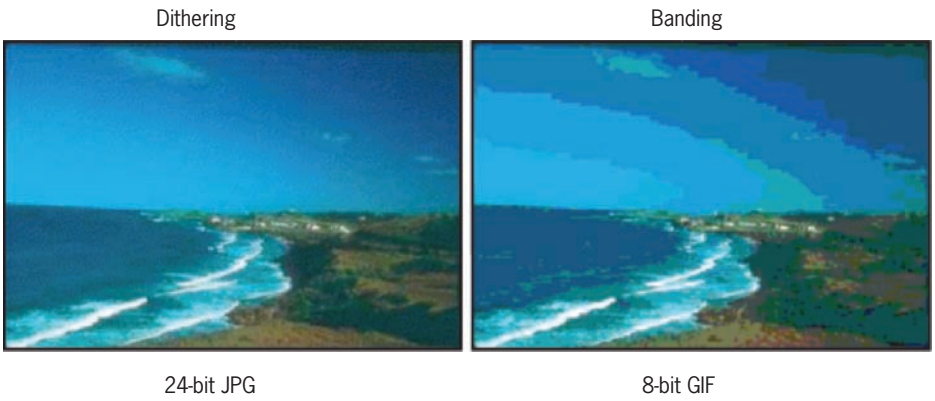


Figure 8-15 24-bit images on an 8-bit display

The JPG file on the left has a lot of dithering in the sky area of the photo, where the browser was forced to mix colors to approximate the existing colors in the image. The GIF file on the right exhibits a different type of color matching called banding. Unlike dithering, **banding** is an effort to match the closest colors from the GIF's palette to the original colors in the photo. When you create a GIF, you can choose whether or not to use dithering. A nondithered image is smaller than one that uses dithering, but the banding may create an unacceptable image. JPGs, when viewed on an 8-bit or 16-bit display, dither to the closest colors. Photos are best saved as JPGs, even when viewed at a lower color depth, because the dithering creates a more acceptable image.

Using the Web Palette

One way to control dithering is to create images that use non-dithering colors. The 216 nondithering colors that are shared by PCs and Macintoshes are called the **Web palette** or **browser-safe colors**. The nondithering palette only applies to GIF or 8-bit PNG, not to 24-bit JPG. Most Web-capable graphics programs include the Web palette colors. If you do create graphics for the Web, you can avoid trouble by using the Web palette as your color palette for all flat color areas of your graphics.

The Web palette, which was once the only color palette acceptable for the majority of monitors, is now becoming less important as higher color depth monitors become the norm. Unless you know specifically that your audience has older monitors, you can forgo using the Web palette.

Creating Web Site Color Schemes

The color scheme used on a Web site can be the result of many factors, including the company's branding colors, designer preferences, and usability studies. Colors convey important

meanings to the user and set the tone for a Web site. Think of the types of colors a designer might choose for a site that promotes ecology and conservation versus a site that celebrates spicy foods. The color choices that come to mind would certainly be different, possibly cool greens and blues for the ecology site, and hot yellows and reds for the spicy foods site. The scheme of colors you choose should work together to create a distinctive look for your site without detracting from your content's legibility. Choosing colors for your site can be difficult, so it helps to have a basic understanding of color theory to help make choices that suit your site's needs.

The study of color theory began with Isaac Newton's series of experiments with prisms published in 1672. Newton found that with a prism, he could separate white light into its component colors: red, orange, yellow, green, blue, and violet. Newton arranged these colors into a wheel that arranged colors logically, as show in Figure 8-16.

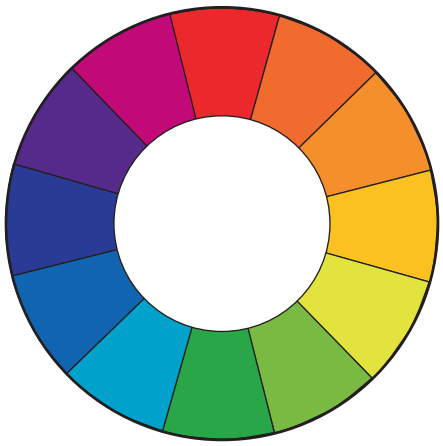
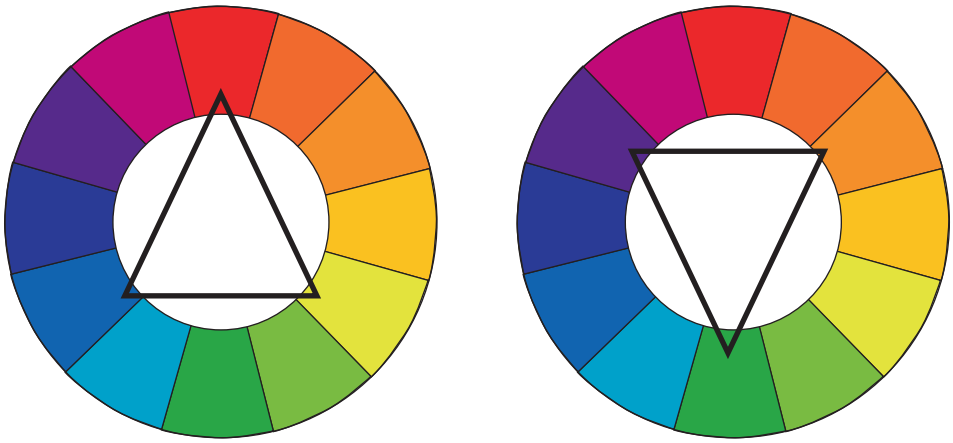


Figure 8-16 Color wheel

In Newton's color wheel, the primary colors—red, yellow, and blue—are arranged opposite their complementary colors; for example, red is opposite green. The primary colors are basic colors of light that cannot be created by mixing other colors. The secondary colors are combinations of primary colors. White, black, and gray are neutral and not included in the wheel. For example, Figure 8-17 shows how the color wheel arranges various relationships between colors.



Primary colors Secondary colors

Figure 8-17 Primary and secondary colors on the color wheel

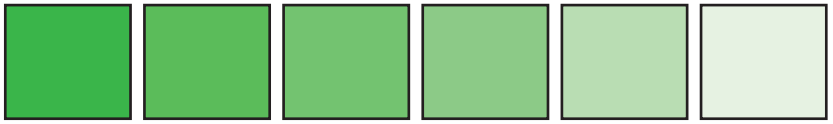
Warm and Cool Colors

The color wheel is often divided into warm and cool colors. The warm colors include colors that are normally seen in daylight or sunsets, reds through yellow, browns, and tan. The cool colors are associated with water, clouds, and overcast days, and include blue through greens and violet. Although these perceptions are culturally dependent, they provide a broad characterization of color usage. The warm colors are generally seen as vivid and energetic, while cool colors are calming and relaxing.

Tints and Shades

In color theory, a pure color is called a **hue**, a color without a tint or shade. If a color is made lighter by adding white, the result is called a **tint**. If black is added, the darker version is called a **shade**. The result of adding these neutral colors to a pure color is shown in Figure 8-18.

Tints: Adding white to a pure hue



Shades: Adding black to a pure hue

Figure 8-18 Tints and shades of colors

Types of Color Schemes

When using color for Web sites, the color wheel can guide your color choices.

Complementary color schemes use the complementary colors that are always arranged opposite of each other on the color wheel. Complementary colors are vivid opposites and do not always go well together, despite their name. Complementary colors are a poor choice for text and backgrounds (for example, yellow text on a blue background) because of their high contrast. Using a complementary color scheme brings high contrast and excitement to your content. See Figure 8-19.

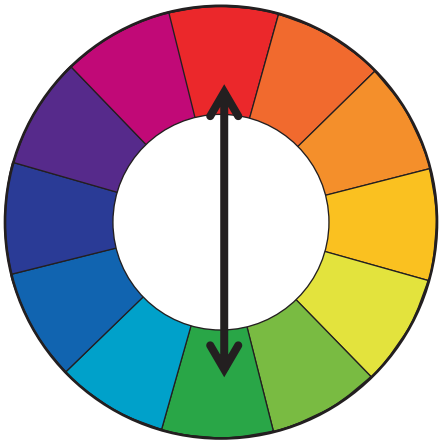


Figure 8-19 Complementary colors

Analogous color schemes use the analogous colors that are located next to each other on the color wheel as shown in Figure 8-20. Analogous colors match well and create designs that are harmonious and pleasing to the eye. One color is usually dominant while the other colors are used to enhance the color scheme. Analogous schemes sometimes need the addition of a more contrasting color to add interest or highlight sections of a layout.

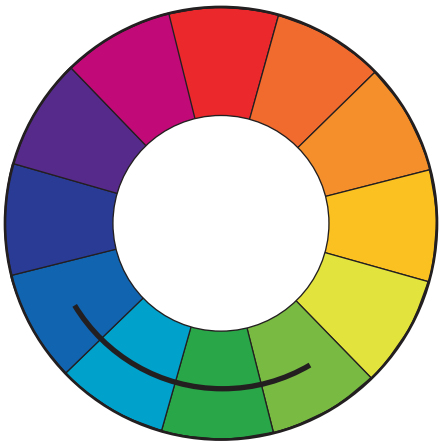


Figure 8-20 Analogous colors

Monochromatic color schemes use tint and shades of a single hue, as shown in Figure 8-21. This scheme looks unified and clean. Monochromatic colors go well together and are easy on the eyes, especially with cool colors. This scheme is a common choice for Web designers who want to create a dignified, understated look. The primary color can be integrated with the neutral colors black white and gray. Like the analogous scheme, it can be difficult to highlight the most important elements because of a lack of color contrast.

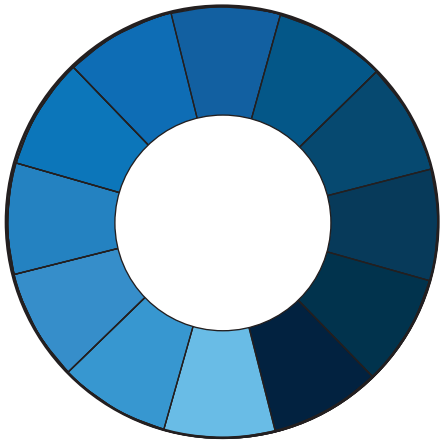


Figure 8-21 Monochromatic colors

The Color Wizard (*www.colorsontheweb.com/colorwizard.asp*) is an easy-to-use color matching tool that can help you choose colors for your Web site designs.

The paletteman Web site (*http://paletteman.com*) lets you choose colors by theme. You can choose up to five colors and see how they interact.

Using Color Wisely

Because of the variable nature of color on the Web, be sure to test the colors you choose, and use restraint when adding color to your design. Remember that colors do not look the same on different monitor brands and operating systems. When used properly, color can enhance the presentation of your information, providing structural and navigation cues to your user. Conversely, poor use of color distracts from your content and can annoy your users. Dark backgrounds, clashing colors, and unreadable links are just a few examples of the unrestrained use of the HTML color attributes that are common on the Web. Just because CSS allows you to easily apply color to any element does not mean that you should apply color haphazardly. Remember that many of your users might have accessibility issues that prevent them from seeing color the way you do. The user's ability to navigate, read, and interact with your content should always determine the choices and use of color in a Web site.

Approximately 7–10 percent of the male population in the United States is color blind. You can test your designs to determine how they will look to users affected by color blindness with one of the following utilities:

www.etre.com/tools/colourblindsimulator

www.vischeck.com/vischeck

Specifying CSS Color Values

In this section, you will learn about the different ways to express color using CSS properties. CSS lets you specify color values in one of three ways:

- Color names
- RGB color values
- Hexadecimal color values

Which color value method should you use? Hexadecimal color values probably should be your first choice because they are supported by all browsers and are the Web's color language. Both hexadecimal and RGB values are more specific and let you express a wider range of color than the color names. Whichever method you choose, make sure to use that method consistently throughout your entire Web site.

Using Color Names

The color name values let you quickly state color using common names. The valid CSS color name values are the 16 basic color names stated in the W3C HTML 4.01 specification, listed in Table 8-5. These are still acceptable for use today.

Color Name	Hex	Color Name	Hex
Aqua	00FFFF	Navy	000080
Black	000000	Olive	808000
Blue	0000FF	Purple	800080
Fuchsia	FF00FF	Red	FF0000
Gray	808080	Silver	C0C0C0
Green	008000	Teal	008080
Lime	00FF00	White	FFFFFF
Maroon	800000	Yellow	FFFF00

Table 8-5 Color Names Recognized by Most Browsers

Although the color names are easy to use, they allow only a small range of color expression. To use a wider variety of available color, you must use a more specific value, such as RGB or hexadecimal.

Using RGB Colors

The RGB color model is used to specify numeric values that express the blending of the red, green, and blue color channels. When you specify RGB values, you are mixing the three basic colors to create a fourth color. Each of the three color channels can be specified in range from 0 to 100%, with 0 representing the absence of the color, and 100% representing the full brilliance of the color. If all three values are set to 0, the resulting color is black, which is the absence of all color. If all three color values are set to 100%, the resulting color is white, which is the inclusion of all colors.

The syntax for specifying RGB is the keyword *rgb* followed by three numerical values in parentheses—the first for red, the second for green, the third for blue. The following rule states a percentage RGB value:

```
p {color: rgb(0%, 100%, 100%);}
```

RGB color values can be specified as an integer value as well. The integer scale ranges from 0 to 255 with 255 equal to 100%. The following rules specify the same color:

```
p {color: rgb(0%, 100%, 100%);} /* percentages */
p {color: rgb(0, 255, 255);} /* integers */
```

Using Hexadecimal Colors

Sometimes you will see colors that you like on a Web site but don't know what the exact color values are. A color picker or eye dropper tool lets you sample a color onscreen so you know the rgb or hexadecimal value. You can download an excellent free color picker tool at *www.iconico.com/colorpic*.

HTML uses hexadecimal numbers to express RGB color values, and you can use them in CSS as well. Hexadecimal numbers are a base-16 numbering system, so the numbers run from 0 through 9, and then A through F. When compared to standard base-10 numbers, hexadecimal values look strange because they include letters in the numbering scheme. Hexadecimal color values are six-digit numbers; the first two define the red value, the second two define the green, and the third two define the blue. The hexadecimal scale ranges from 00 to FF with FF equal to 100%. Hexadecimal values are always preceded by a pound sign (#). The following rules specify the same color:

```
p {color: #00ffff;} /* hexadecimal */
p {color: rgb(0%, 100%, 100%);} /* percentages */
p {color: rgb(0, 255, 255);} /* integers */
```

Understanding Element Layers

The color and background properties you will learn about in this chapter let you control three different layers of any element. You can imagine these layers as three individual pieces of tracing paper laid over each other to complete the finished Web page. Each layer is transparent until you add a color or an image. These are the three layers listed in order from back to front:

- *Background color layer*—The back or bottom layer, specified by the background-color property
- *Background image layer*—The middle layer, specified by the background-image property
- *Content layer*—The top layer; this is the color of the text content, specified by the color property

Figure 8-22 shows the three layers and their order from front to back. The background color layer (colored sky blue) lies behind all of the other layers. The background image layer displays the balloon image, which overlays the background color. The top layer contains the content. Notice that the content layer overlays both the background image and background color layers.

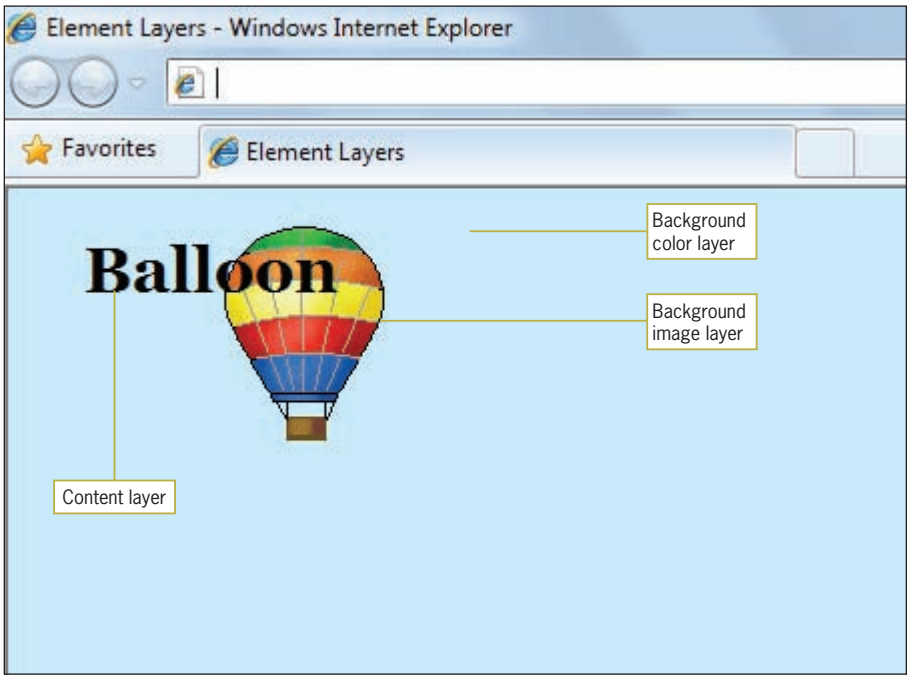


Figure 8-22 Element layers

Controlling Color Properties with CSS

In this section you will use Cascading Style Sheet properties to control the following color characteristics:

- Specifying color values
- Setting default text color
- Changing link colors
- Specifying background color
- Setting the page background color
- Creating a text reverse

Specifying Color Values

Color property description	
Value:	<color>
Initial:	depends on browser
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

The color property lets you specify the foreground color of any element on a Web page. This property sets the color for both the text and the border of the element unless you have specifically stated a border color with one of the border properties (see Chapter 6).

The value for the color property is a valid color keyword or numerical representation, either hexadecimal or RGB (described earlier in the "Using RGB Colors" section). The following style rules show the different methods of specifying the same color:

```
p {color: blue;}              /* color name */
p {color: #0000ff;}           /* hexadecimal value */
p {color: rgb(0,0,255);}      /* RGB numbers */
p {color: rgb(0%,0%,100%);}   /* RGB percentages */
```

Figure 8-23 shows an <h1> element with the color set to red (hexadecimal #f90000). By default, the element's border is the same color as the element content.

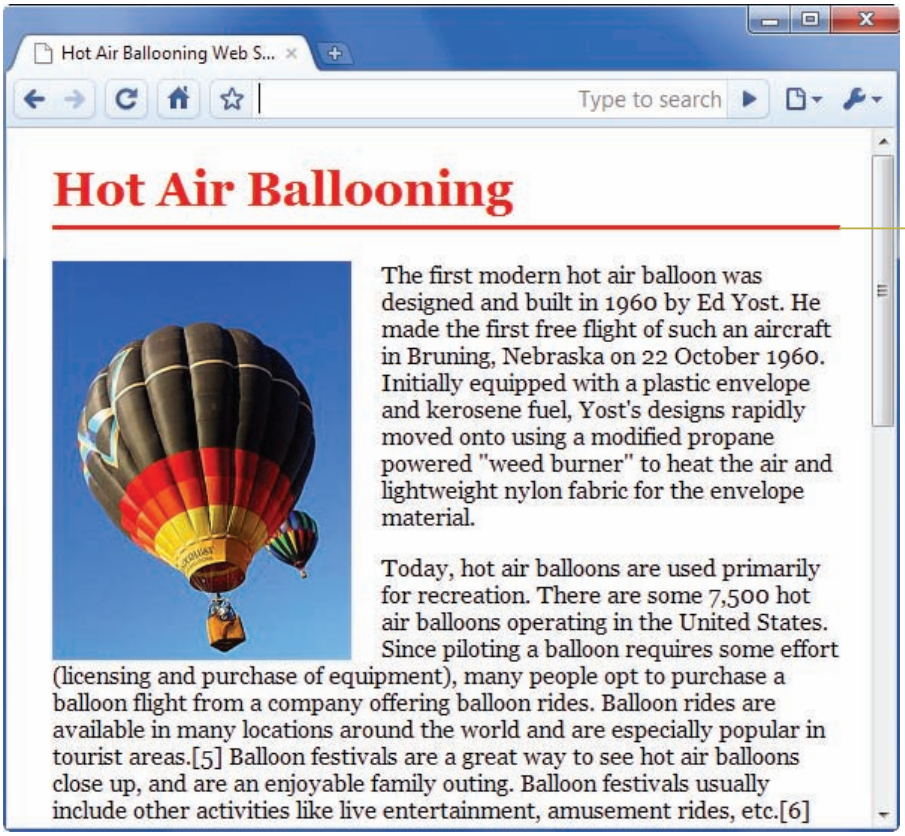


Figure 8-23 Element border defaults to the text color

Here is the style rule for the heading. Notice that the border color is not specified, so the element's border is the same color as the element text.

```
h1 {
   color: #f90000;
   border-bottom: 3px solid;
   padding-bottom: 6px;
}
```

Setting the Default Text Color

Color is inherited from parent to child elements. If you set the color for <body>, all elements on the page inherit their color from the <body> element, effectively setting the default text color for the entire Web page. The following rule sets the color for the <body> element:

```
body {
   color: #006633;}
```

Changing Link Colors

You can change the colors of hypertext links by using the link pseudo-classes:

- *link*—The unvisited link color; the default is blue.
- *active*—The active link color; this is the color displayed when the user points to a link and holds down the mouse button. The default is red.
- *visited*—The visited link color; the default is purple.

The following code shows the link pseudo-classes in use:

```
a:link {color: #cc0033;} /* new links are red */
a:active {color: #000000;} /* active links are black */
a:visited {color: #cccccc;} /* visited links are green */
```

Figure 8-24 shows a text-based navigation bar where the links have been colored red to match the design of the heading.

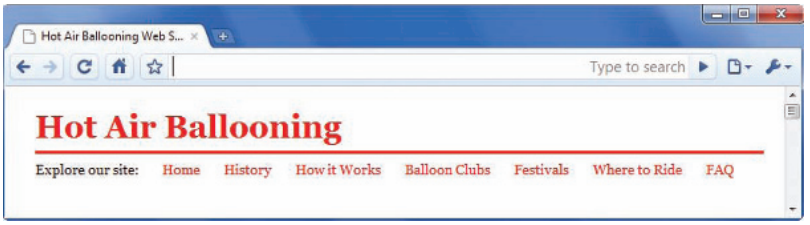


Figure 8-24 Changing link colors

Remember to place your link pseudo-class in the following order:

1. Link
2. Visited
3. Hover
4. Active

Refer to Chapter 4 for more information on link pseudo-classes.

The familiar blue (for new links) and purple (for visited links) colors are one of the most recognizable navigation cues for users visiting your site. Keep in mind that some users might have sight disabilities, such as color blindness, that could prevent them from seeing your Web pages in the way you intend. However, many sites do change their links to match their design color scheme. Changing link colors is acceptable as long as you maintain color consistency and preserve the contrast between the new and visited link colors to provide a recognizable difference to the user.

Specifying Background Color

The background-color property lets you set the background color of any element on a Web page.

The background color includes any padding area (explained in Chapter 6) that you have defined for the element. Figure 8-25

shows an <h1> element with background color and padding. The style rule looks like this:

```
h1 {
   color: #f90000;
   background-color: #fec893;
   border-bottom: 3px solid;
   padding-top: 20px;
   padding-bottom: 6px;
   padding-left: 20px;
}
```

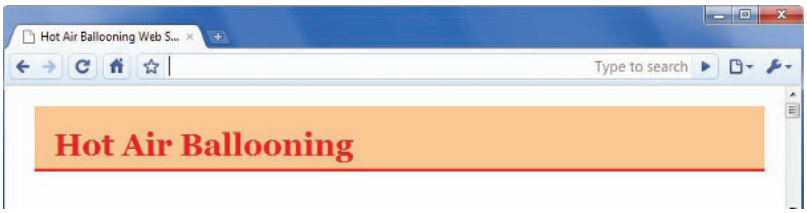


Figure 8-25 Background color and padding

The background-color property can be applied to both block-level and inline elements. To apply a background image color to inline text, use the element to select the text. The following style rule selects a span element to apply a background color with 4 pixels of padding:

```
span.bgcolor {
   background-color: #fec893;
   padding: 4px;
}
```

This span element with class="bgcolor" selects the words "Hot Air" in the heading:

```
<span class="bgcolor">Hot Air</span> Ballooning
```

The result is shown in Figure 8-26.

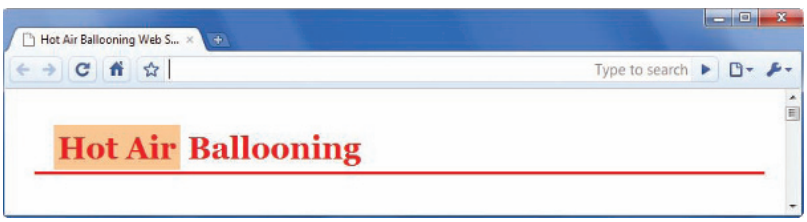


Figure 8-26 Inline element with a background color

Setting the Page Background Color

To set the page background color, use *body* as the selector. This sets the background color for the content area of the Web page. By default, the background color of any element is transparent. Therefore, all elements show the page background color unless the background-color property is specifically stated. The following rule sets a background color for the <body> element as shown in Figure 8-27.

```
body {background-color: #c5f0ff;}
```

Notice in Figure 8-27 that the navigation links at the top of the page have a white background. This is because the element that contains the navigation links has a background color set to white.

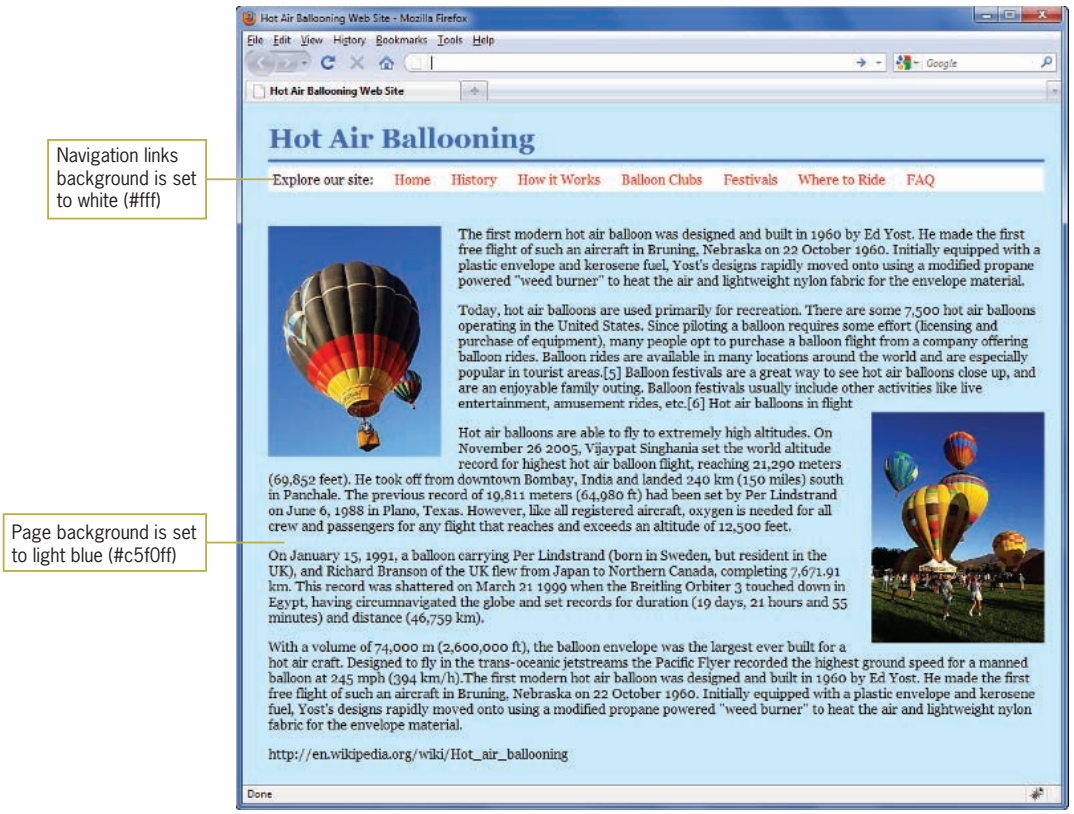


Figure 8-27 Page background color

It is always a good practice to include a page background color because some users might have a default background color that is different from the color you chose in your design. Even if you plan

on a white page background, you can never be sure that all users have their default set to white, so include the background-color property rather than relying on the user's settings.

Creating a Text Reverse

A reverse is a common printing effect where the background color, which is normally white, and the text color, which is usually black, are reversed. On the Web you can do this in your choice of color. Reverses are usually reserved for headings rather than the regular body text. You can easily create a reverse with a style rule. The following rule sets the background color of the <h1> element to red and the text color to white:

```
h1 {
   background-color: #f90000;
   padding: 10px;
   color: #fff;
}
```

The element padding is set to 10 pixels to increase the background color area. Figure 8-28 shows the result of the style rule.

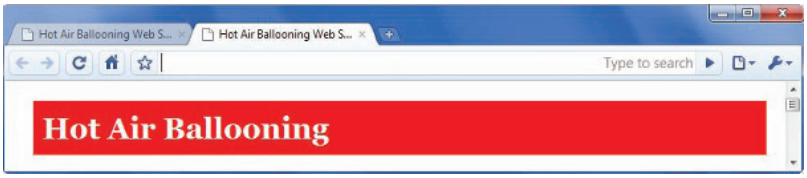


Figure 8-28 Reverse text in a heading

Controlling Background Images with CSS

In this section, you will use Cascading Style Sheet properties to control the following background characteristics:

- Specifying a background image
- Creating a page background
- Specifying background repeat
- Creating a vertical and horizontal repeat
- Creating a nonrepeating background image
- Specifying background position
- Positioning repeating background images

Specifying a Background Image

background-image property description	
Value:	<url>
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The background-image property lets you specify which image to display. Other CSS background properties control how the image is displayed.

With standard HTML, the only behavior of background images is to tile completely across the browser background. This is also the default behavior with the CSS background-image property. Figure 8-29 shows a document with an image tiled across the background.

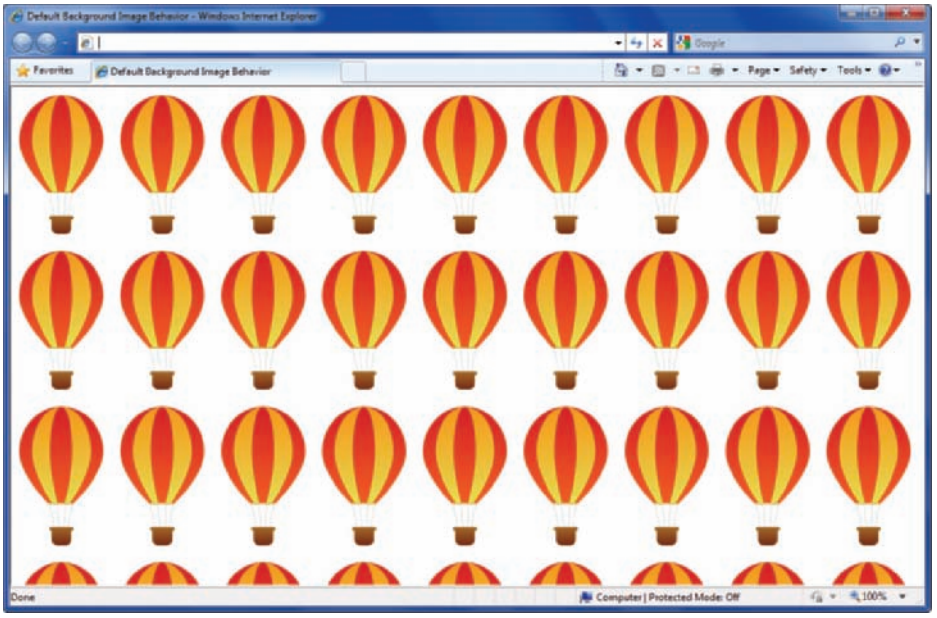


Figure 8-29 Default background image behavior

The background image from this example is shown in Figure 8-30. It is tiled repeatedly both vertically and horizontally.

Figure 8-30 Individual background image

In Figure 8-29, the background would obviously detract from the legibility of any Web page text. When choosing page backgrounds, keep the legibility of your text in mind. Avoid overly busy and distracting backgrounds that make your content difficult to read.

Specifying the Background Image URL

To specify a page background image, use the <body> element as the selector, because <body> is the parent element of the content area. To use an image in the background, you must specify the relative location of the image file in the style rule. CSS has a special notation for specifying a URL, as shown in Figure 8-31.

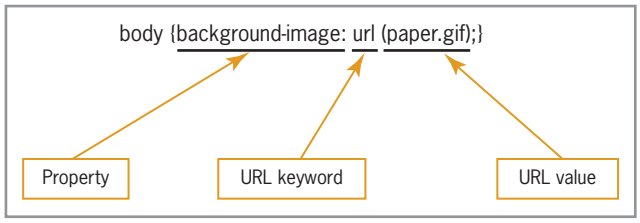


Figure 8-31 URL value syntax

If you are using external style sheets, the URL of the background image is relative to the location of the style sheet, not the HTML file to which the style sheet is applied.

Creating a Page Background

To tile an image across the entire background of the Web page, use body as the selector, as shown in the following rule. This is the style rule that was used to create the background in Figure 8-32.

```
body {background-image: url(clouds.jpg);}
```

In this example, a seamless background graphic tiles repeatedly across the page background, and behind the wrapper division that contains the page content. This technique lets you frame your content on the left and right margins with a background color that integrates with your design. This also fills the browser window no matter the user's resolution, so your pages are always framed by an active part of the design, rather than passive screen space.

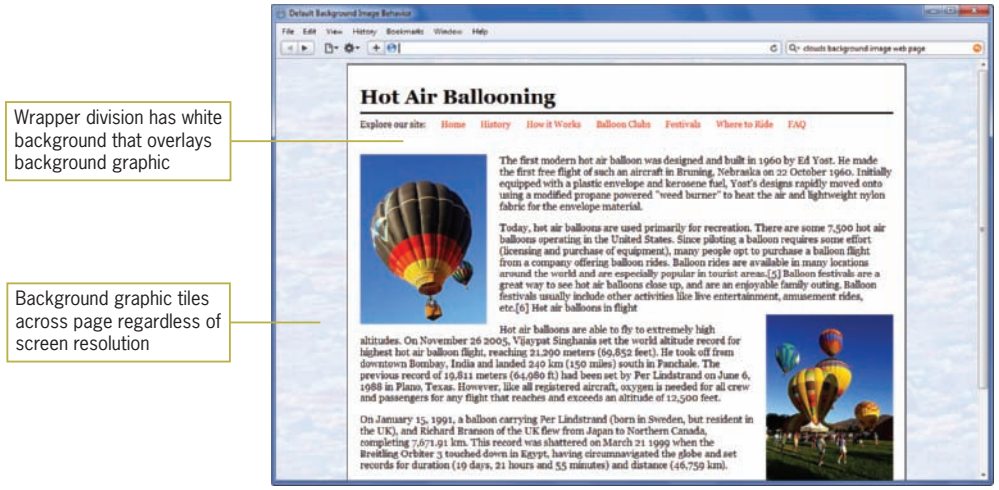


Figure 8-32 Repeating page background behind wrapper division

Figure 8-33 shows the seamless graphic that was used to create the page background in Figure 8-32.

Figure 8-33 Seamless image used to create continuous background

Specifying Background Repeat

background-repeat property description	
Value:	repeat | repeat-x | repeat-y | no-repeat | inherit
Initial:	repeat
Applies to:	all elements
Inherited:	no
Percentages:	N/A

The background-repeat property lets you control the tiling of background images across the document or element background.

A background image must be specified for this property to work, so you always use the background-image property with the background-repeat property. Table 8-6 lists the background-repeat values.

Value	Background Image Behavior
repeat	The image is repeated across the entire background of the element; this is the default behavior
repeat-x	The image is repeated across the horizontal (x) axis of the document only
repeat-y	The image is repeated across the vertical (y) axis of the document only
no-repeat	The image is not repeated; only one instance of the image is shown in the background

Table 8-6 Background-Repeat Property Values

Creating a Vertical Repeat

The repeat-y value of the background-repeat property lets you create a vertical repeating background graphic. Figure 8-33 shows an example of this effect. The background graphic shown in Figure 8-34 is a 200-pixel wide by 50-pixel high JPG file.

Figure 8-34 Background graphic used to create background column

This property lets you easily create columns with image or color backgrounds because the graphic is repeated vertically. You can

then align content or division elements over the background image columns as shown in Figure 8-35.

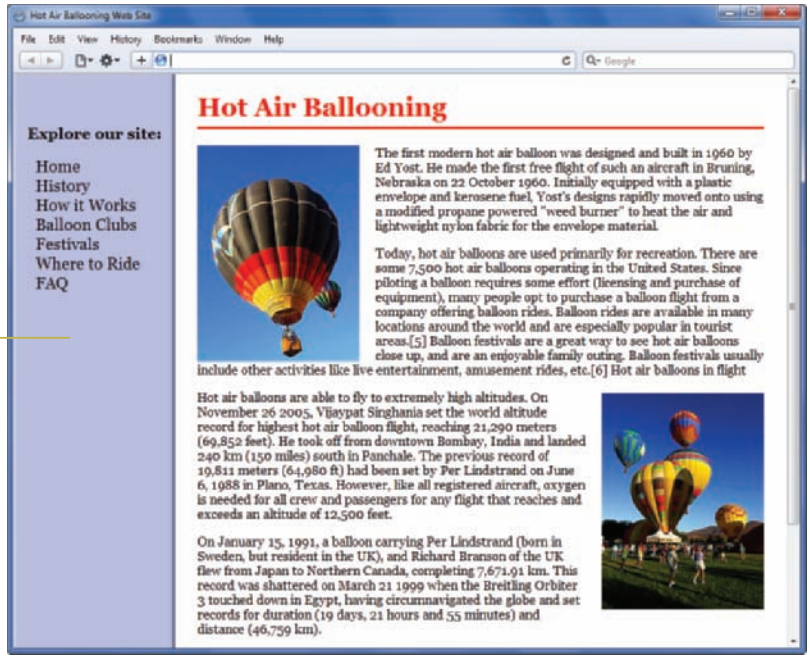


Figure 8-35 Vertical repeating background image

The Web page in this figure is a 2-column layout as you saw in Chapter 7. The navigation content is contained in a division that is the same width as the background graphic behind it. The background graphic is repeated only on the y-axis to create a vertical column. The style rule for the background uses body as the selector with background-repeat set to repeat-y:

```
body {
   background-image: url(column.jpg);
   background-repeat: repeat-y;
}
```

Creating a Horizontal Repeat

The repeat-x value of the background-repeat property lets you create a horizontal repeating background graphic. Figure 8-33 shows an example of this effect. The background graphic shown in Figure 8-36 is a 50-pixel wide by 110-pixel wide graphic.

Figure 8-36 Background graphic used to create background banner

This property lets you easily create a background banner with a graphic that is repeated horizontally as shown in Figure 8-37.

Figure 8-37 Horizontal repeating background image

The style rule for the background uses body as the selector with background-repeat set to repeat-x:

```
body {
   background-image: url(header.jpg);
   background-repeat: repeat-x;
   }
```

Creating a Nonrepeating Background Image

The no-repeat value of the background-repeat property lets you create a single instance of an image in the background. This is a great way to add images to your site that appear consistently as part of your layout or branding.

The following style rule shows the use of the no-repeat value:

```
body {
   background-image: url(balloon_sm.jpg);
   background-repeat: no-repeat;}
```

The background position property is normally used with a non-repeated image to position it properly. Figure 8-38 shows a single balloon image centered at the bottom of a division element.

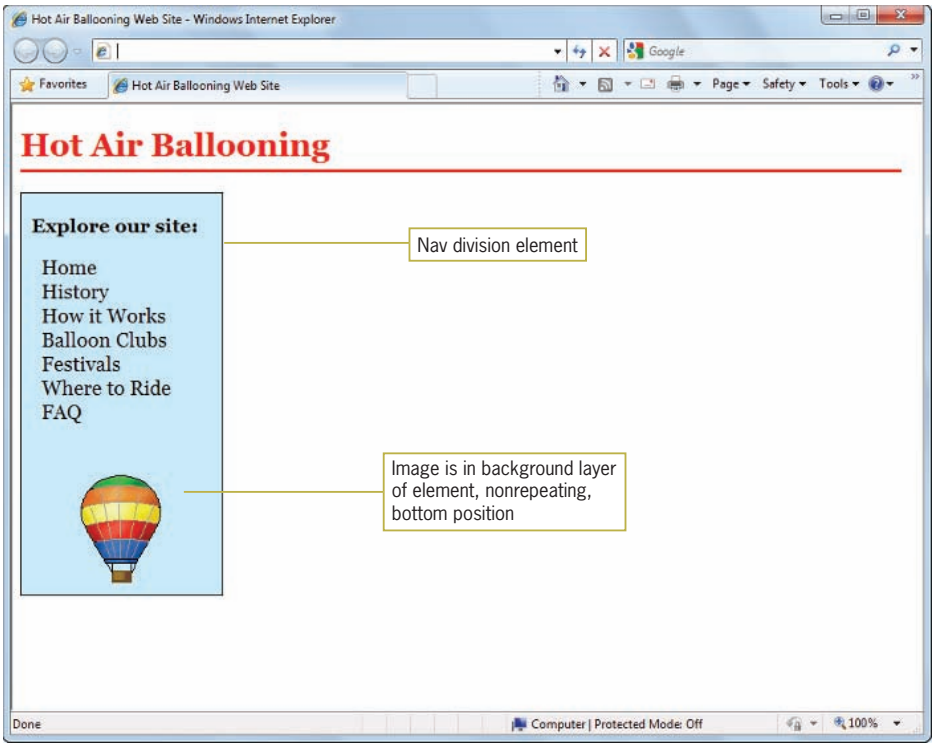


Figure 8-38 Nonrepeating background image positioned in column

Specifying Background Position

background-position property description	
Value:	[[<percentage> | <length>]{1,2} | [[top | center | bottom] || [left | center | right]]
Initial:	0% 0%
Applies to:	block-level and replaced elements
Inherited:	no
Percentages:	refer to the size of the box itself

The background-position property lets you use three types of values: percentage, length, or keywords. Table 8-7 lists the values

and their meanings. Figure 8-39 shows the keyword positions in the element box and their equivalent percentage values.

You can use the keywords in Table 8-7 alone (*left*) or in combination (*left top*) to position the background image. Figure 8-39 shows the nine keyword positions and their percentage equivalents. The keywords can be used interchangeably, so the values *left top* and *top left* are the same.

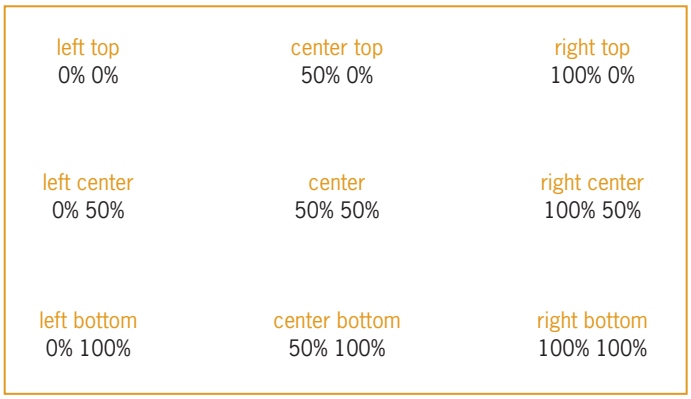


Figure 8-39 Keyword and percentage background positions

Value	Background Image Behavior
percentage	The percentage values are based on the starting point of the upper-left corner of the containing element's box. The first percentage value is horizontal; the second is vertical. For example, the value *45% 30%* places the background image 45% from the left edge and 30% from the top edge of the containing box.
length	Length values work in much the same way as percentages, starting from the upper-left corner of the element's containing box. The first length value is horizontal; the second is vertical. For example, the value *100px 200px* places the background image 100 pixels from the left edge and 200 pixels from the top edge of the containing box
keywords	The keywords are: • left • right • center • top • bottom

Table 8-7 Background-position Property Values

Positioning Repeating Background Images

You can also position images that repeat on either the horizontal or vertical axis of the Web page. The following style rule positions the vertical repeating background image along the right side the element:

```
#right {
   background-image: url(rightgradient.gif);
   background-repeat: repeat-y;
   background-position: right;}
```

Figure 8-40 shows the four different alignments of repeating images. For repeat-y, the default is left. For repeat-x, the default is top.

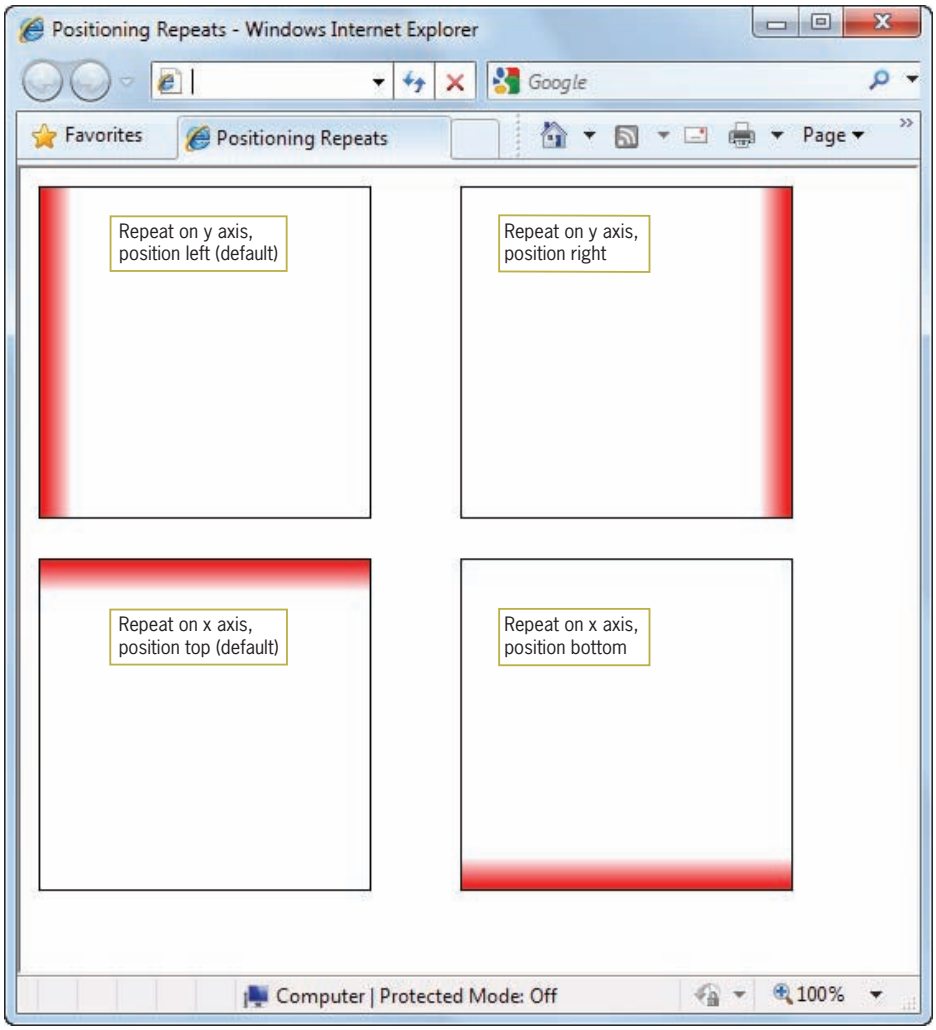


Figure 8-40 Four positions for repeats

Once again, these repetitive borders are composed from a single image, in this case a small gradient box that was rotated with an image-editing program for the four different positions. The graphic is shown in Figure 8-41.

Figure 8-41 This gradient image is rotated as necessary to create the repeats

Chapter Summary

To create an engaging, accessible, and informative Web site, you must use graphics wisely. Keep the following points in mind:

- The four popular file formats for the Web are GIF, JPG, PNG, and SVG. The first three formats compress images to create smaller files. Unless you choose the appropriate file format, your image will not compress and appear as you expect. As a vector graphics format, SVG graphics are scalable and cross-platform compatible.
- Your computer monitor displays color by mixing the three basic colors of light: red, green, and blue (RGB). Colors vary widely from one monitor to another, based on both the user's preferences and the exact brand of equipment.
- Reduce image size to the appropriate dimensions for a Web page.
- The color scheme you choose for a Web site should work together to create a distinctive look without detracting from your content's legibility. Use hexadecimal values when specifying colors for your Web site. Color names are not always the best way to specify color values because of their variable nature.
- Use the color property to set foreground colors for elements. Remember that the element border defaults to the element color unless you specifically state a border color.
- Background colors affect any padding areas in the element. They can be applied to both block-level and inline elements.

- Choose background images that do not detract from the legibility of your content. Use the background-repeat and background-position properties to control the appearance of images in the background.
- Test your work on different browsers and computing platforms, because they render colors differently. Test at different color depths as well.

Key Terms

analogous color scheme—A color scheme that uses colors located next to each other on the color wheel.

aspect ratio—The the ratio of width to height in an image or shape.

banding—An effect created in GIF images when browsers try to match the closest colors from the GIF's palette to the original colors in the image.

browser-safe colors—The 216 colors shared by PCs and Macintoshes. These colors are displayed properly across both platforms without dithering. These are now becoming less important as higher color depth monitors become the norm.

color depth—The amount of data used to create color on a display. The three common color depths are 8-bit, 16-bit, and 24-bit. Not all displays support all color depths.

complementary color scheme—A color scheme that uses colors that are arranged opposite of each other on the color wheel.

dithering—This color-mixing process occurs when a browser encounters a color on a Web page that it does not support. The browser is forced to mix the color. The resulting color may be grainy or unacceptable. To avoid dithering, work with browser-safe colors.

Graphics Interchange Format (GIF)—A file format designed for online delivery of graphics. The color depth of GIF is 8-bit, allowing a palette of no more than 256 colors. The GIF file format excels at compressing and displaying flat color areas, making it the logical choice for line art and graphics with simple colors.

hue—A pure color in color theory.

interlacing—The gradual display of a graphic in a series of passes as the data arrives in the browser. Each additional pass of data creates a clearer view of the image until the complete image is displayed. You can choose an interlacing process when you are creating GIFs.

Joint Photographic Experts Group (JPG or JPEG)—A file format, commonly shortened to JPG, designed for the transfer of photographic images over the Internet. JPGs are best for photos and images that contain feathering, complex shadows, or gradations.

monochromatic color scheme—A color scheme that uses tint and shades of a single hue.

Portable Network Graphics (PNG)—A graphics file format for the Web that supports many of the same features as GIF.

raster graphics—Images represented pixel-by-pixel for the entire image. GIFs and JPGs are raster formats.

Scalable Vector Graphics (SVG)—A language for describing two-dimensional graphics using XML. SVG files can contain shapes such as lines and curves, images, text, animation, and interactive events.

shade—A color made darker by adding black.

tint—A color made lighter by adding white.

vector graphics—Images represented as geometrical formulas, as compared with a raster graphics format, which represents images pixel by pixel for the entire image. SVG is a vector graphic format. Vector graphics are scalable and cross-platform compatible.

Web palette—The 216 colors shared by PCs and Macintoshes. These colors display properly across both platforms without dithering.

Review Questions

1. What are the three image file formats you can use on a Web site?

2. Which file formats support 24-bit color?

3. How many colors does GIF support?
4. What is the browser-safe palette?
5. What is lossless file compression?
6. Which file formats support transparency?
7. What are the drawbacks of using animated GIFs?
8. Explain lossy image compression.
9. What image characteristics can you control using the JPG format?
10. What are some options for acquiring images for your site?
11. Which image format should you use for a two-color company logo?
12. Which image format should you use for a photograph?
13. What three attributes should you always include in the image tag? Why?
14. How many layers can you work with when designing pages?
15. What are the three different ways to express color values in CSS?
16. How is the default border color of an element determined?
17. What are the three special selectors that let you change link colors?
18. To what type of elements can you apply a background color?
19. What is the default background image behavior?

Hands-On Projects

1. Practice using the CSS float property.
    a. Download an image from the Online Companion Web site, or find an image of your own.
    b. Add text around the image. Experiment with the float property and its values to view the way text wraps.
    c. Test the work in multiple browsers to verify that the text wraps consistently.

2. Practice using the CSS margin property attributes with images.
    a. Download an image from the Online Companion Web site, or find an image of your own.
    b. Add text around the image. Experiment with the margin property to add white space around the image.
    c. Test the work in multiple browsers to verify that the text spacing is consistent.

3. Practice using width and height image attributes.
    a. Download an image from the Online Companion Web site, or find an image of your own.
    b. Build a simple page that contains text and multiple images. Do not include the width and height attributes in the <img> tag.
    c. With the images turned off in your browser, view the page.
    d. Add the appropriate width and height information to the <img> tag for each image.
    e. Again, turn the images off in your browser and view the page. Note the differences between the two results and the way your layout is affected.

4. In this project, you add an image and color information to a Web page. The code you will add to the file appears in blue.

   a. Copy the **wildflowers.html** file and the **daisy.jpg** file from the Chapter 08 folder provided with your Data Files to the Chapter08 folder in your work folder. (Create the Chapter 08 folder, if necessary.)

   b. Start your text editor, and open the file **wildflowers.html**.

   c. Add an <img> element to the page immediately after the opening <p> tag, as shown in the following code in blue text:

```
<html>
<head>
<title>Growing Wildflowers</title>
</head>
<body>
<h1>Growing Wildflowers</h1>
(p)<img src="daisy.jpg" /> Lorem ipsum dolor sit
    amet, consectetuer adipiscing elit, sed diem
    nonummy nibh euis mod tincidunt ut lacreet
    dolore magna aliguam erat volutpat.

...body text...

    adipiscing elit, sed diem nonummy nibh euismod
    tincidunt ut lacreet dolore magna aliguam erat
    volutpat. Ut wisis enim ad minim veniam, quis
    nostrud exerci tution ullamcorper suscipit
    lobortis nisl ut aliquip ex eacommodo consequat.
</p>
</body>
</html>
```

   d. Save the file and view it in the browser. It should look like Figure 8-42.

Figure 8-42 Adding an image to a Web page

e. Add attributes to the image to provide size and alternate text information. The image width and height are both 100 pixels. The alt and title attributes can contain any text you choose to describe the image, such as *daisy image*. The following code fragment shows the attribute additions:

```
<img src="daisy.jpg" width="100" height="100"
alt="daisy image" title="The Wild Daisy" />
```

f. Wrap the text around the image by adding a CSS style rule to the image. Use the style attribute with the float property set to *left* as shown in the following code fragment:

```
<img src="daisy.jpg" width="100" height="100"
alt="daisy image" title="The Wild Daisy"
style="float: left;" />
```

g. Save the file and view it in the browser. When you view the file it looks like Figure 8-43.

Figure 8-43 Floating the image to the left of text

h. Adjust the right margin of the image by adding a margin-right property to the style attribute. Set the measurement value to 20px, as shown in the following code:

```
<img src="daisy.jpg" width="100" height="100"
     alt="daisy image" title="The Wild Daisy"
     style="float: left; margin-right: 20px;" />
```

i. Save the file and view it in the browser. It should look like Figure 8-44.

Figure 8-44 Adding a right margin to the image

j. Add a style attribute to the <h1> element to change the color to a forest green. The hexadecimal code is 006633. The following code fragment shows the <h1> element with the style attribute.

```
<h1 style="color: #006633">Growing Wildflowers</h1>
```

k. Finish the page by setting the background color to a light green. Add a style attribute to the body element, and set the background color to light green, hexadecimal value 99cc99, as shown in the following code fragment:

```
<body style="background-color: #99cc99">
```

l. Save the file and close the editor. Then view the finished page in the browser. It should look like Figure 8-45, with a deep green heading and light green page background. The complete code for the page follows.

Figure 8-45 Completed wildflowers.html Web page

```
<html>
<head>
<title>Growing Wildflowers</title>
</head>
<body style="background-color: #99cc99">
<h1 style="color: #006633">Growing Wildflowers</h1>
```

```
<p><img src="daisy.jpg" width="100" height="100"
    alt="daisy image" title="The Wild Daisy"
    style="float: left; margin-right: 20px;"> Lorem
    ipsum dolor sit amet, consectetuer adipiscing
    elit, sed diem nonummy nibh euis mod tincidunt
    ut lacreet dolore magna aliguam erat volutpat.

...body text...

    adipiscing elit, sed diem nonummy nibh euismod
    tincidunt ut lacreet dolore magna aliguam erat
    volutpat. Ut wisis enim ad minim veniam, quis
    nostrud exerci tution ullamcorper suscipit
    lobortis nisl ut aliquip ex eacommodo consequat.
</p>
</body>
</html>
```

5. In this project, you have a chance to apply some of the background properties you learned about in this chapter. As you work through the steps, refer to Figure 8-47 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Copy the **mars.html** and **pattern1.jpg** files from the Chapter08 folder provided with your Data Files to the Chapter08 folder in your work folder. Then open **mars.html** in your HTML editor.

   b. In your browser, open **mars.html**. When you open the file, it looks like Figure 8-46.

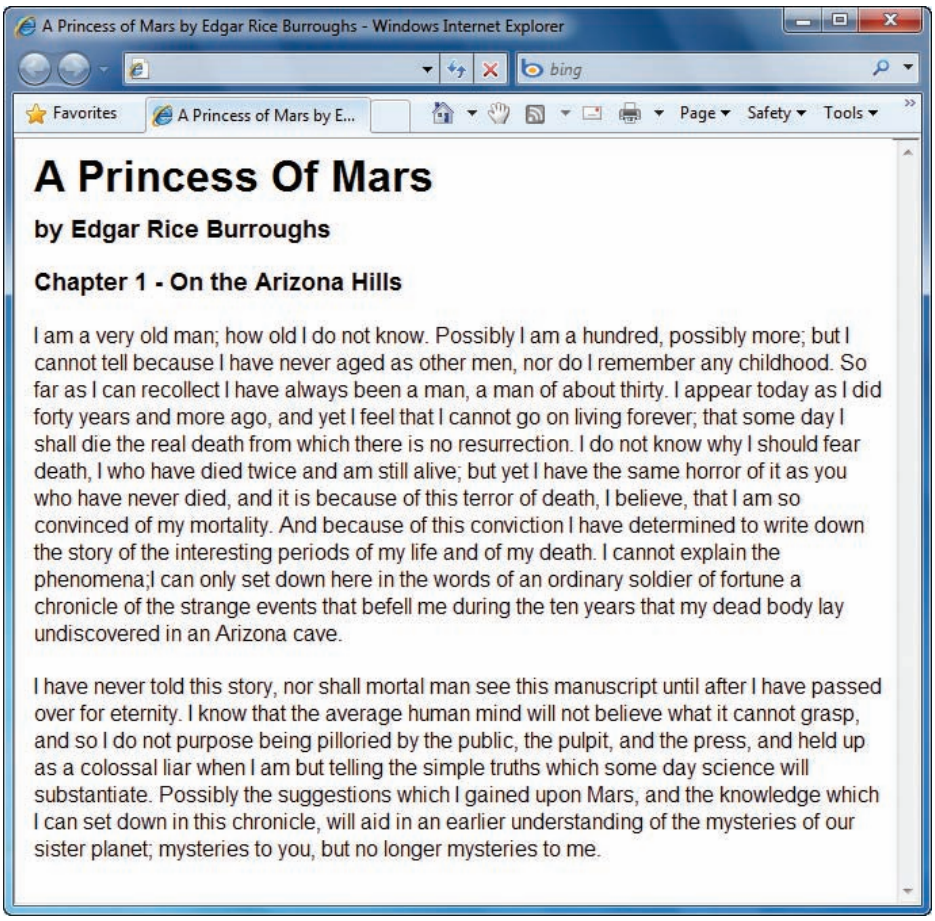


Figure 8-46 Beginning mars.html Web page

c. Examine the code. Notice the <style> section of the file. It currently contains a style rule that sets the font family and line height for the text. The complete code for the page follows:

```
<html>
<head>
<title>A Princess of Mars by Edgar Rice Burroughs</
  title>
<style type="text/css">
body   {font-family: sans-serif;
       line-height: 1.25em;}
</style>
</head>
<body>
<h1>A Princess Of Mars</h1>
<h3>by Edgar Rice Burroughs</h3>
```

```
<h3>Chapter 1 - On the Arizona Hills</h3>
<p>I am a very old man; how old I do not know.
   Possibly I am a hundred, possibly more; but I
   cannot tell because I have never aged as other
   men, nor do I remember any childhood. So far
   as I can recollect I have always been a man, a
   man of about thirty. I appear today as I did
   forty years and more ago, and yet I feel that
   I cannot go on living forever; that some day I
   shall die the real death from which there is no
   resurrection. I do not know why I should fear
   death, I who have died twice and am still alive;
   but yet I have the same horror of it as you who
   have

...body text...

   Possibly the suggestions which I gained upon
   Mars, and the knowledge which I can set down
   in this chronicle, will aid in an earlier
   understanding of the mysteries of our sister
   planet; mysteries to you, but no longer mysteries
   to me.</p>
</body>
</html>
```

d. In your HTML editor, start by setting the background color for the Web page. The finished design uses a brown background. Write a style rule that uses body as the selector and sets the background-color property to #cc6633 (reddish brown):

```
<style type="text/css">
body   {font-family: sans-serif;
        line-height: 1.25em;
        background-color: #cc6633;}
</style>
```

e. Next, build the style for the <h1> element, which is a text reverse. Use h1 as the selector, specify a background color of #663300 (dark brown) and a text color of #ffffff (white). Add padding of .25em:

```
h1 {color: #ffffff; background-color: #663300;
    padding: .25em;}
```

f. Add a background image for the Web page (**pattern1.jpg**). Add the background-image property to the existing style rule for the <body> element, because you want to apply the background image to the entire Web page.

```
body {font-family: sans-serif;
     line-height: 1.25em;
     background-color: #cc6633;
     background-image: url(pattern1.jpg);}
```

g. When you test the new style rule you added in Step 5f, you see that the background image tiles across the entire background of the Web page. You want to restrict the tiling of the background graphic to the left margin of the browser. To accomplish this, use the background-repeat property set to repeat-y. Add this property to the existing style rule:

```
body {font-family: sans-serif;
     line-height: 1.25em;
     background-color: #cc6633;
     background-image: url(pattern1.jpg);
     background-repeat: repeat-y;}
```

h. The background now repeats correctly on the left side of the browser window, but the content text is illegible against it. To fix this problem, add a margin-left property for all of the elements that contain text: <h1>, <h2>, <h3>, and <p>. Specify a value of 125px, as shown in the following rule:

```
h1, h2, h3, p {margin-left: 125px;}
```

i. The finished code for the style sheet follows. Figure 8-47 shows the completed Web page.

```
<html>
<head>
<title>A Princess of Mars by Edgar Rice Burroughs
 </title>
<style type="text/css">
body {font-family: sans-serif;
     line-height: 1.25em;
     background-color: #cc6633;
     background-image: url(pattern1.jpg);
     background-repeat: repeat-y;}
h1   {color: #ffffff;
     background-color: #663300;
     padding: .25em}
h1, h2, h3, p {margin-left: 125px;}
</style>
</head>
```

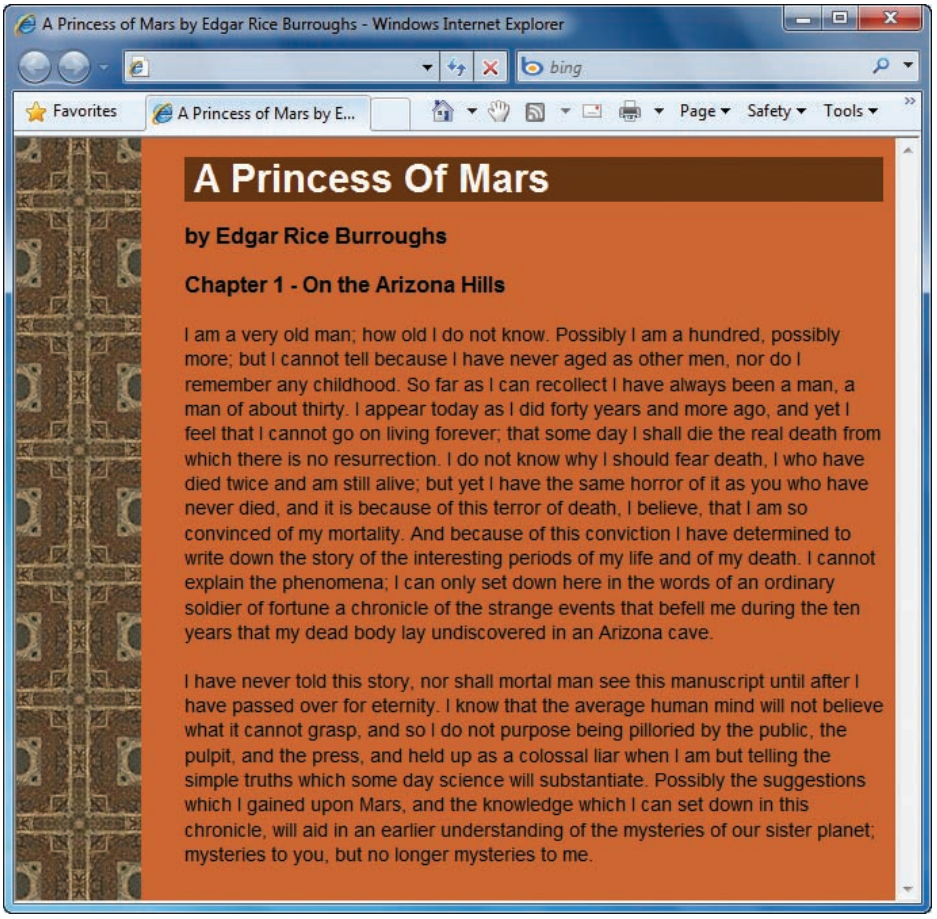


Figure 8-47 Completed mars.html Web page

6. Browse the Web and choose a site that you feel exhibits positive use of color, in both content and backgrounds. Write a short design critique that describes how the use of color enhances the legibility of the site and improves user access to information.

7. Browse the Web and choose a mainstream (not amateur) site that can benefit from a change in color scheme. Look for problems with legibility of text over background colors, use of nonstandard linking colors, and so on. Write a short essay that describes the changes you would implement to improve the use of color on the site.

Individual Case Project

Gather or create the graphics to use on the different pages of your site. These include any banner, navigation, section, or identifying graphics. Add these graphics to the test pages of your site. Test the images in multiple browsers to make sure they are displayed properly.

Think about the different color requirements for your content, and decide how you can enhance the legibility of the content. Can color communicate information about the structure of your information?

Determine the color choices for your Web site. Pick the colors for text, the background color in tables, and page backgrounds.

Establish graphics standards for your Web site, including but not limited to the following:

- Decide whether you will use a standard amount of white space around each graphic.
- Determine exactly which img attributes should be included in all <img> tags.
- Formulate a standard for all alt and title attributes.
- Formulate a basic set of image standards for your site. Use this as the display standard for testing your graphics.
- Determine colors of links and visited links.

Write a short standards document that can be provided to anyone contributing to the site.

Team Case Project

Work with your team to decide on the graphics and color choices for your project Web site. These include any banner, navigation, section, or identifying graphics, and the colors for text, the background color in tables, and page backgrounds.

You may need to bring sample graphics or mock-up HTML pages to present your ideas on these characteristics to your team members.

Establish graphics standards for your Web site, including but not limited to the following:

- Decide whether you will use a standard amount of white space around each graphic.
- Determine exactly which img attributes should be included in all <img> tags.
- Formulate a standard for all alt and title attributes.
- Formulate a basic set of image standards for your site. Use this as the display standard for testing your graphics.
- Determine colors of links and visited links.

Write a short standards document that can be submitted to the instructor and provided to the team members.

After you have reached a general consensus, go back to work on the page template you adopted in Chapter 7. Create more finished mock-ups of your page design. Trade the page layout examples with your team members. Look for unifying characteristics that give your site a unique identity. Make sure the colors and graphics flow though the different page levels on the site. Work towards smooth transitions between your pages. You want all the pages to exhibit a graphic identity that connects them together.

CHAPTER 9

Site Navigation

When you complete this chapter, you will be able to:

- Create usable navigation
- Build text-based navigation
- Use graphics for navigation and linking
- Use lists for navigation
- Build horizontal navigation bars
- Build vertical navigation bars
- Using background color and graphics to enhance navigation
- Create hover rollovers

The free-flowing nature of information in a nonlinear hypertext environment can be confusing to navigate. Help your users find content easily rather than making them hunt through a maze of choices. Let your users know where they are at all times and where they can go within your Web site. In this chapter, you learn to build user-focused navigation to accomplish these goals.

Creating Usable Navigation

Webopedia (*www.webopedia.com*) defines hypertext as "a system in which objects (text, pictures, music, programs, and so on) can be creatively linked to each other.... You can move from one object to another even though they might have very different forms." Hypertext was envisioned in the 1960s by Ted Nelson, who described it as nonsequential writing in his book *Literary Machines*. Nelson's basic idea of connecting content through hypertext linking influenced the creators of the Web. With hypertext-linked content, users can traverse information in any order or method they choose, creating their own unique views.

Hypertext is a distinctly different environment in which to write and structure information. In traditional paper-based media, users navigate by turning pages or referring to a table of contents or an index separate from the information they are reading. In a hypertext document, users can connect instantly to related information. The hypertext forms of traditional navigation devices, such as tables of contents and cross-references, can be displayed constantly alongside related content. The user can explore at will, jumping from one point of interest to another. Of course, the ease of navigation depends on the number of links and the context in which they were added by the hypertext author.

In HTML, hyperlinks are easy to create and add no extra download time when they are text based. When you are planning your site navigation, do not skimp on navigation cues, options, and contextual links. You can use the CSS style properties you have learned about to create attractive navigation elements. If you use graphics for navigation, remember to keep your navigation graphics simple and reuse the same graphics throughout your Web site. Once the navigation graphics are loaded in the user's cache, the server does not have to download them again. Use an alternate set of text links in the event that the user cannot or will not view your graphics and to meet accessibility guidelines. You will learn more about text links later in this chapter.

Effective navigation includes providing not only links to other pages in the Web site, but also cues to the user's location. Users should be able to answer the following navigation questions:

- Where am I?
- Where can I go?
- How do I get there?
- How do I get back to where I started?

To allow users to answer these questions, provide the following information:

- The current page and the type of content they are viewing
- Where they are in relation to the rest of the Web site
- Consistent, easy-to-understand links
- Alternatives to the browser's Back button that let users return to their starting point

Locating the User

Figure 9-1 shows a page from the National Archives Web site (*www.archives.gov*) that displays a number of user-orienting features.

Figure 9-1 Providing user location cues

The navigation cues on this page offer many options without disorienting the user. A search option lets the user search the entire site. A linked **breadcrumb path** at the top of the page shows the user's location within the site hierarchy. Users can click any of the links in the path to move through the content structure. This location device is especially effective in guiding users who may have arrived at this page from somewhere outside this Web site. The section heading identifies the current section, and the links on the left let the user jump to related content pages. Using these navigation devices, users can choose to jump directly to a page, search for information, or move back up through the information hierarchy.

Limiting Information Overload

Many Web sites tend to present too much information at one time. Lengthy files that require scrolling or have arrays of links and buttons can frustrate and overwhelm the user. You can limit information overload in the following ways:

- *Create manageable information segments*—Break your content into smaller files, and then link them together. Provide logical groupings of choices. Keep a flat hierarchy. A good rule to follow is that users should not have to click more than two or three times to get to the information they desire.
- *Control page length*—Do not make users scroll through never-ending pages. Long files also can mean long downloads. Provide plenty of internal links to help users get around, and keep the pages short. You can judge your page length by pressing the Page Down key; if you have to press it more than two or three times to move from the top to the bottom of your page, break up the content.
- *Use hypertext to connect facts, relationships, and concepts*—Provide contextual linking to related concepts, facts, or definitions, letting the users make the choices they want. Know your material, and try to anticipate the user's information needs.

Activity: Building Navigation Structures

Text-based linking often is the most effective and accessible way to provide navigation on your site. In the following set of steps, you will link a series of sample Web pages using text-based navigation techniques. Figure 9-2 shows the structure of the collection of sample HTML documents that you will use, including the Home page, Table of Contents page, Site Map page, and individual Chapter pages.

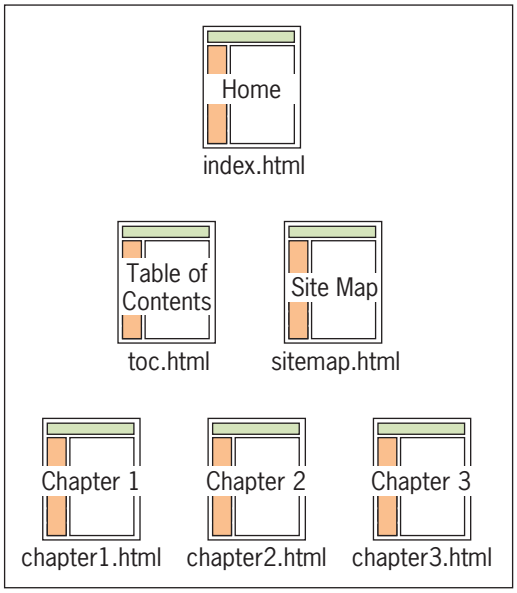


Figure 9-2 Sample file structure

This book's Online Companion Web site contains all the HTML files for the sample Web site illustrated in Figure 9-2.

In the hypertext environment, the user should be able to select links in the table of contents to jump to any document in the collection. In the following steps, you will add a variety of linking options that produce different paths through the information. The focus for these steps is the Table of Contents page, toc.html, and how it relates to the rest of the content in the collection. You will also add navigation options to the individual chapter pages. The Index and Site Map pages are included to complete the sample Web site and will be target destinations for some of the links you will build. You can then apply these linking techniques to your own Web site projects.

To complete the steps in this chapter, you need to work on a computer with a browser and an HTML editor or a simple text editor, such as Notepad or SimpleText.

To prepare for linking the Web pages:

1. Copy the following files from the Chapter09 folder provided with your Data Files:
   - index.html
   - toc.html
   - sitemap.html
   - chapter1.html

- chapter2.html
- chapter3.html
- styles.css

2. Save the files in the Chapter09 folder in your work folder using the same filenames. (Create the Chapter09 folder, if necessary.) Make sure you save all the files in the same folder.

In the following set of steps, you use common HTML coding techniques. If necessary, review your basic HTML knowledge before proceeding with these instructions.

Linking with a Text Navigation Bar

The Table of Contents page must link to the other main pages of the Web site, allowing users to go directly to the pages they want. You can achieve this by adding a simple text-based navigation bar.

To build the navigation bar:

1. From the Chapter09 folder in your work folder, open the file **toc.html** in your browser. It should look like Figure 9-3.

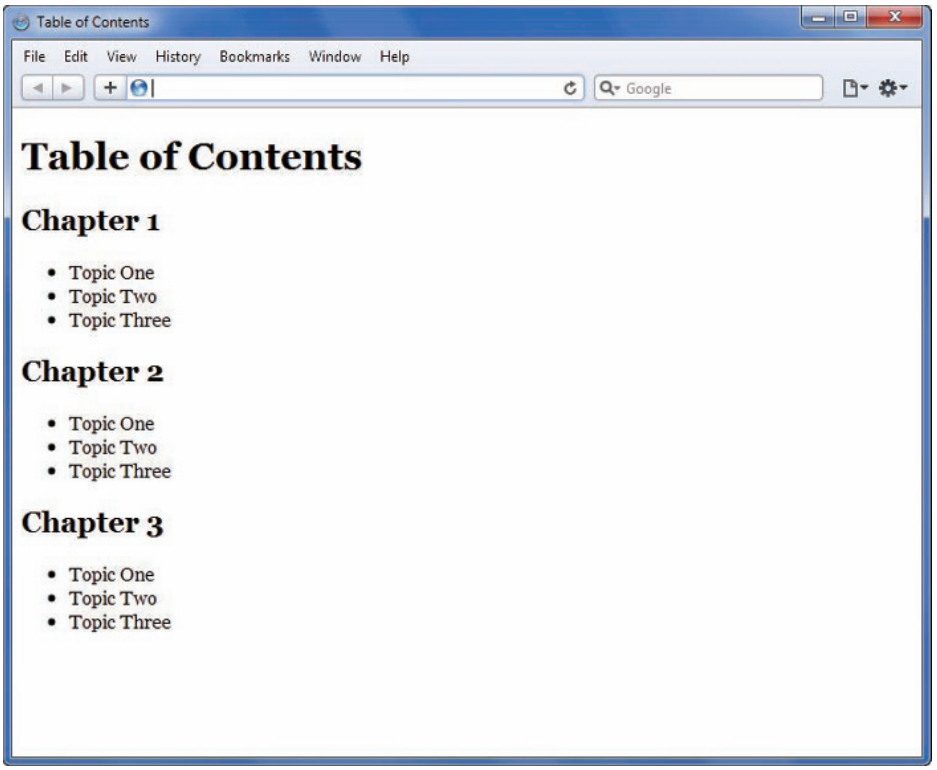


Figure 9-3 Original HTML file

2. Open the file in your HTML editor, and examine the code. Notice that the file contains a link to an external style sheet that controls the basic styles for the site. The complete code for the page follows:

```
<!DOCTYPE html>
<html>
<head>
<title>Table of Contents</title>
<meta content="text/html; charset=utf-8" http-equiv=
   "Content-Type" />
<link href="styles.css" rel="stylesheet" type="text/css" />

</head>
<body>
<h1>Table of Contents</h1>

<h2>Chapter 1</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
<h2>Chapter 2</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
<h2>Chapter 3</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
</body>
</html>
```

In this set of steps, you use the id selector to provide an identifying name for the element. You can use the id name to apply CSS styles to a section of a Web page. Refer to Chapter 4 to read more about the id selector.

3. Add a <p> element to place the navigation bar immediately following the opening <body> tag. Set the id attribute to *headernav* as shown in the following code. (The new code you should add appears in blue in this step and the following steps.)

```
<body>
<p id="headernav"> </p>
```

4. Add text within the new <p> element as shown in the following code.

```
<p id="headernav">Home | Table of Contents | Site Map</p>
```

5. Add <a> tags with href attributes that link to the home page and the site map.

```
<p id="headernav">
<a href="index.html">Home</a> | Table of Contents |
<a href="sitemap.html">Site Map</a></p>
```

6. Add a span element with a class attribute to contain the Table of Contents text. Because this is the Table of Contents page, the text "Table of Contents" is not a hypertext link but is bold to designate the user's location. The code looks like this:

```
<p id="headernav">
<a href="index.html">Home</a> |
<span class="current">Table of Contents</span> |
<a href="sitemap.html">Site Map</a></p>
```

7. Open the stylesheet file **styles.css** in your editor. When you open the file it looks like this:

```
/* Principles of Web Design 5th ed. */
/* style sheet for linking activity in Chapter 9 */


body {font-family: georgia, serif;}
li {line-height: 125%;}
p {margin: 0px 20px 0px 30px}
```

8. In styles.css, add a style for the id *headernav* that specifies a width, automatic margins, padding, a border, and text alignment. Use a comment to describe the style rule.

```
/* Navigation Header */
#headernav {
   margin-left: auto;
   margin-right: auto;
   padding: 10px;
   border: solid 1px black;
   width: 300px;
   text-align: center;
}
```

9. Add another style for the *current* class that indicates which page the user is currently viewing.

```
/* Current Page Indicator */
.current {
   font-weight: bold;
}
```

10. Save and close the style sheet file, and then view the Table of Contents page in your browser. It should look like Figure 9-4. Test your hypertext links in the navigation bar to make sure they point to the correct pages.

11. Add this navigation bar to all of the sample pages. Remember to change the links and placement of the element to reflect the current page. For example, the Site Map page would have the text "Site Map" contained in the and links to the Table of Contents and Home pages. The chapter pages (chapter1.html, chapter2.html, and chapter3.html) do not need a span element, just links to the Home, Table of Contents, and Site Map pages.

12. Save **toc.html** in the Chapter09 folder of your work folder, and leave it open in your HTML editor for the next steps.

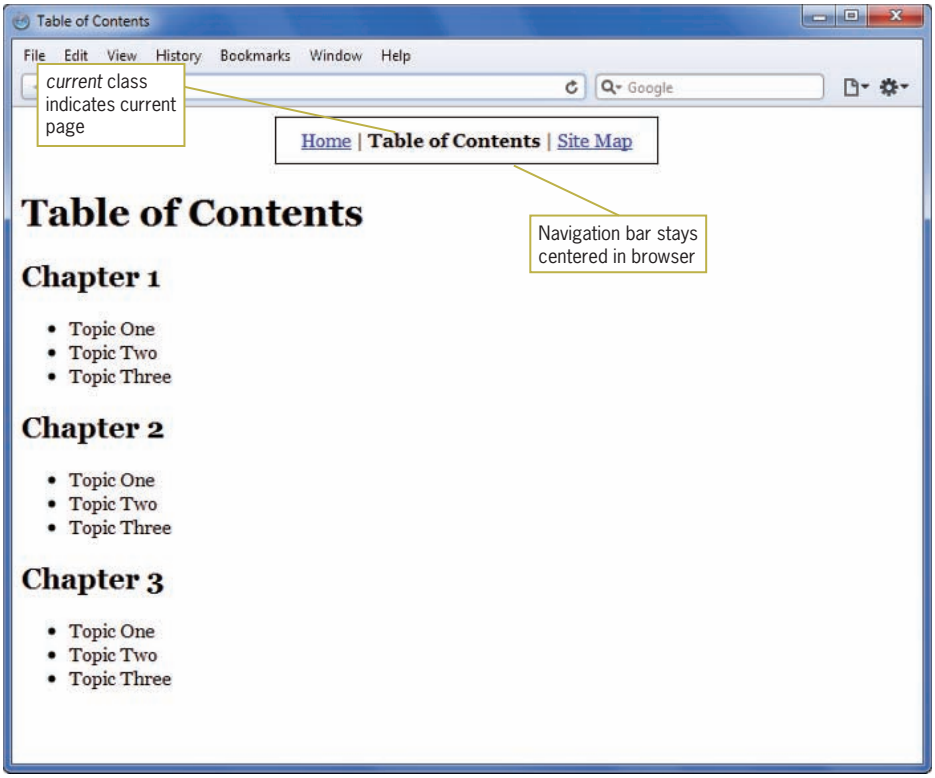


Figure 9-4 Adding a text-based navigation bar

Linking to Chapter Pages

While the navigation bar lets users access the main pages in the Web site, the table of contents lets users access the exact content pages they want. The Table of Contents page therefore needs links to the individual chapter files in the Web site. In this set of steps,

you will add links to the individual chapter files listed in the table of contents.

To build page links:

1. Continue working in the file **toc.html**. Add the following <a> element around the text "Chapter 1":

```
<h2><a href="chapter1.html">Chapter 1</a></h2>
```

2. Add similar <a> elements around the text "Chapter 2" and "Chapter 3" that point to the files chapter2.html and chapter3.html, respectively.

3. Save **toc.html** and view the finished Table of Contents page in your browser. It should look like Figure 9-5. Test your hypertext links to make sure they point to the correct page.

As this example shows, remember always to make <a> the innermost set of tags to avoid extra space in the hypertext link.

This linking method lets the users scroll through the table of contents to scan the chapters and topics and then jump to the chapter they want. The link colors—by default, blue for new and purple for visited—allow users to keep track of which chapters they already have visited.

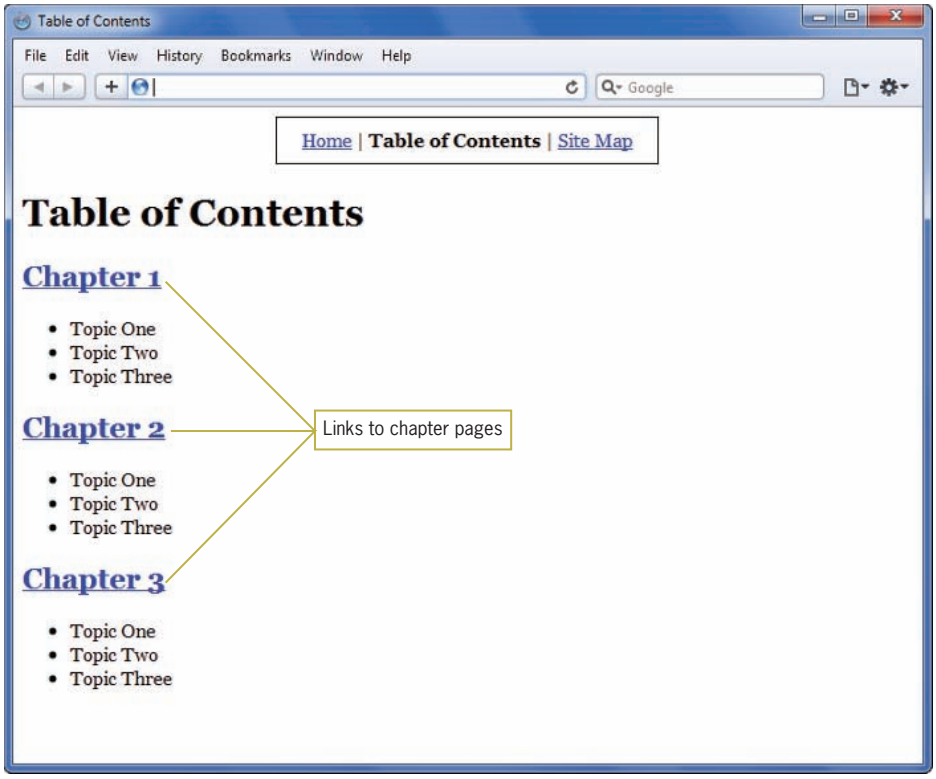


Figure 9-5 Adding chapter links

Adding Internal Linking

In addition to linking to external documents, you also can add internal links for navigating within the table of contents itself. In the Table of Contents page illustrated in Figure 9-5, you will add a Back to Top link that lets users return to the top of the page when they reach the bottom of the page.

This requires two <a> anchor elements: one uses the name attribute to name a **fragment identifier** in the document; the other targets the fragment name in the href attribute.

To add an internal link:

1. Continue working with the file **toc.html** in your editor. Add a new <a> element at the top of the page, immediately after the <body> tag. Add a name attribute, and set the value to *top* as shown.

   ```
   <body>
   <a name="top"></a>
   <p id="headernav">
   <a href="index.html">Home</a> |
   <span class="current">Table of Contents</span> |
   <a href="sitemap.html">Site Map</a></p>
   ```

   Notice that this <a> element is empty. The name attribute identifies this location in the document as "top." You can then refer to this name as an href target elsewhere in the document. The value of the name attribute can be any combination of alphanumeric characters that you choose.

2. Add an <a> element at the bottom of the page, after the listing for Chapter 3 and just before the closing </body> tag. Reference the target fragment *top* by using the number sign (#) in the href attribute, as shown in the following code.

   ```
   <h2><a href="chapter3.html">Chapter 3</a></h2>
      <ul>
       <li>Topic One</li>
       <li>Topic Two</li>
       <li>Topic Three</li>
      /ul>
   <a href="#top">Back to Top</a>
   </body>
   ```

3. Add a <p> element around the <a> element, and set the id attribute to *footer* to identify the division. This will come in handy if you decide to add a style to the footer later.

   ```
   <p id="footer"><a href="#top">Back to Top</a></p>
   ```

4. Save the **toc.html** file, and view the Table of Contents page in your browser. Resize your browser to show only a portion of the page, as shown in Figure 9-6.

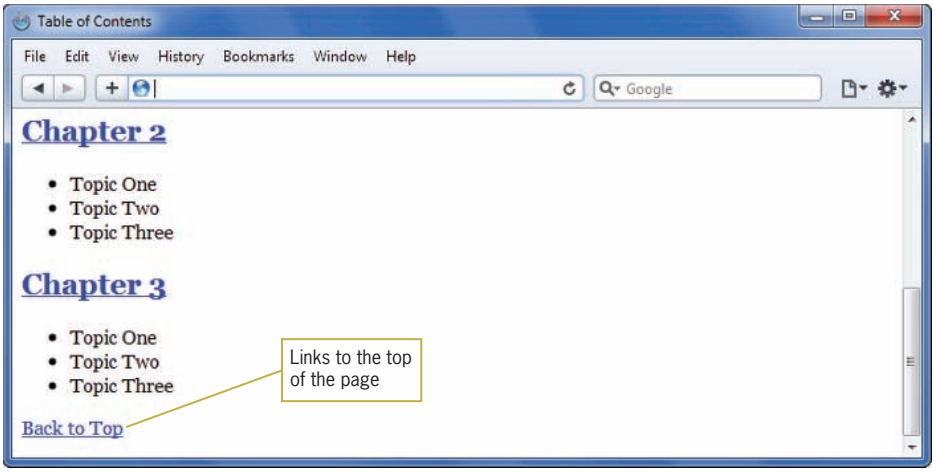


Figure 9-6 Adding a Back to Top link

5. Test the link to make sure it opens the browser window at the top of the page, as shown in Figure 9-7.

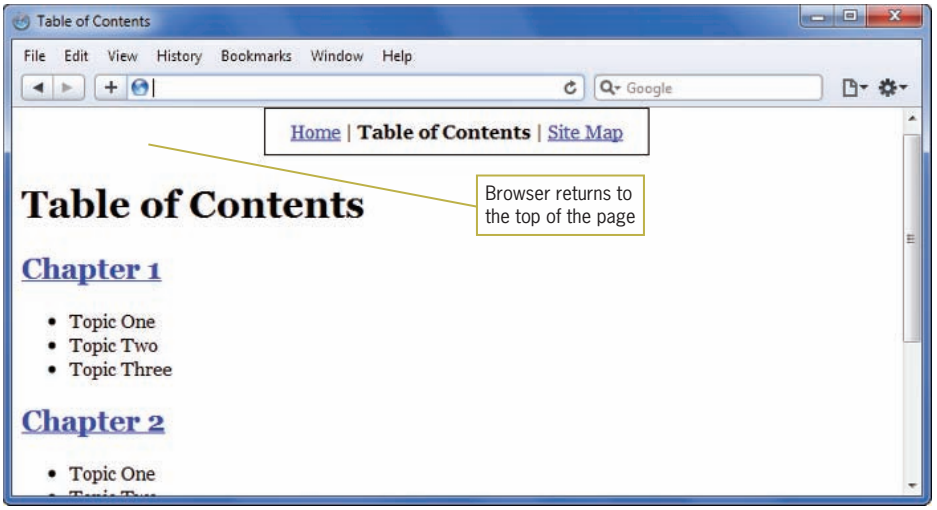


Figure 9-7 Results of testing the Back to Top link

Adding a Page Navigation Bar

You can use additional fragment identifiers in the table of contents to add more user-focused navigation choices. Figure 9-8 shows the addition of a page navigation bar.

When users click one of the linked chapter numbers, they jump to the specific chapter information they want to view within the table of contents further down the page.

To add a page navigation bar:

1. Continue working with the file **toc.html** in your HTML editor. Add a <p> element immediately after the <h1> element as shown. Set the id attribute to *pagenav* as shown in the following code.

```
<h1>Table of Contents</h1>
<p id="pagenav"> </p>
```

2. Add text to the <p> element as shown in the following code.

```
<p id="pagenav">
Jump down this page to Chapter... 1 | 2 | 3</p>
```

3. Add <a> elements around each chapter number within the <p>. Insert href attributes that point to a named fragment for each chapter.

```
<p id="pagenav" style="text-align: center;">
Jump down this page to Chapter...
<a href="#chapter1">1</a> |
<a href="#chapter2">2</a> |
<a href="#chapter3">3</a></p>
```

4. Add the name attribute to each chapter's <a> element. These are the fragment names you referred to in Step 3. The following code shows the new name attributes in the <a> element for each chapter.

```
<h2><a href="chapter1.html" name="chapter1">Chapter 1</a>
</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
<h2><a href="chapter2.html" name="chapter2">Chapter 2</a>
</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
<h2><a href="chapter3.html" name="chapter3">Chapter 3</a>
</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three></li>
   </ul>
```

5. Save the **toc.html** file in the Chapter09 folder in your work folder, and then close the file. View the Table of Contents page in your browser. Resize your browser to show only a portion of the page, as shown in Figure 9-8.

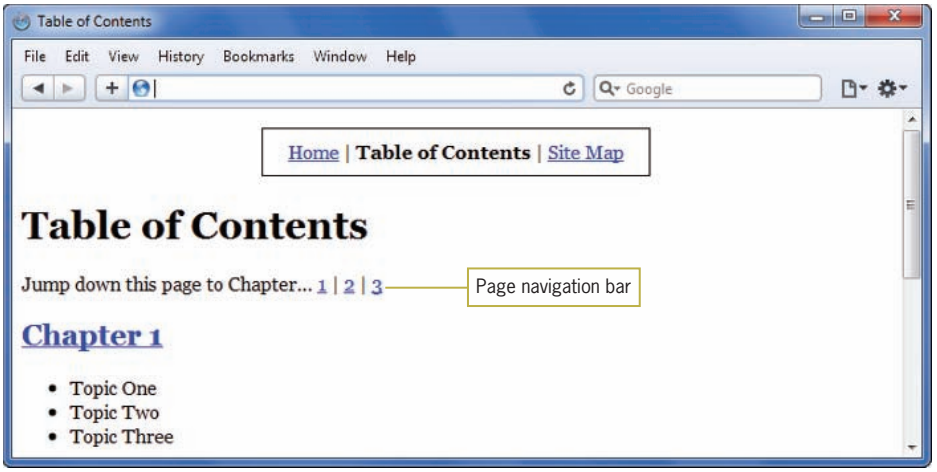


Figure 9-8 Adding a page navigation bar

6. Test the navigation bar links by selecting a chapter number and making sure the browser window opens to the correct place in the file. Figure 9-9 shows the result of selecting the Chapter 2 link.

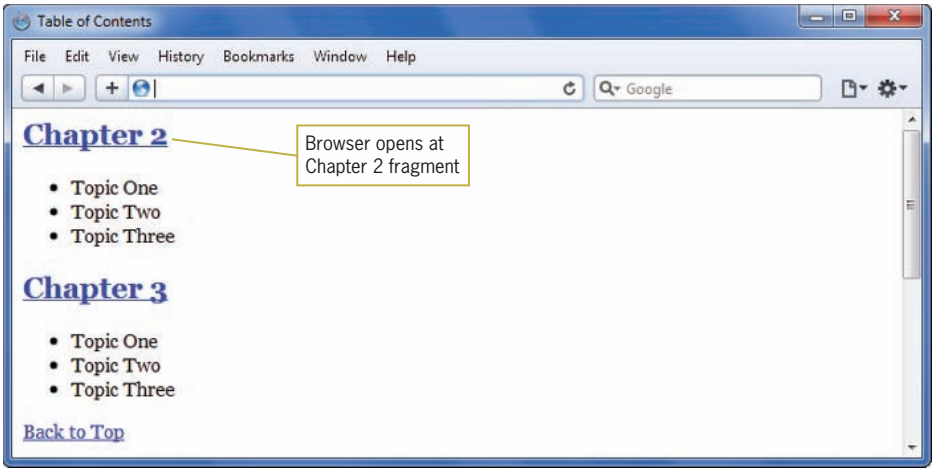


Figure 9-9 Results of testing the page Chapter 2 link

Linking to External Document Fragments

Now that you have completed internal linking in the table of contents, reexamine how the table of contents is linked to each chapter file. Currently, each chapter has one link in the table of contents; users click the chapter link, and the browser opens the chapter file at the top. However, each chapter also contains multiple topics, which can be linked as external fragments. You can let users jump from the table of contents to the exact topic they want within each chapter. This requires adding code to both the Table of Contents page and each individual chapter page.

To add links to external fragments:

1. In your HTML editor, open the file **chapter1.html** from the Chapter09 folder in your work folder.
2. Find the topic headings within the file. For example, the following code creates the topic heading for Topic 1:

   ```
   <h2>Topic 1</h2>
   ```

3. Add an <a> element around the text "Topic1." Set the name attribute value to *topic1* for this heading, as shown in the following code.

   ```
   <h2><a name="topic1">Topic 1</a></h2>
   ```

4. Now add the same code to each of the other topic headings in the file, using *topic2* and *topic3* as the name attribute values for the Topic 2 and Topic 3 headings, respectively.
5. Save **chapter1.html** in the Chapter09 folder of your work folder, and then close the file.
6. In your HTML editor, open the files **chapter2.html** and **chapter3.html** from the Chapter09 folder of your work folder. Repeat Steps 2 through 5 for both files, adding appropriate name attribute values for each topic in each file.
7. In your HTML editor, open the file **toc.html** from the Chapter09 folder of your work folder. (This is the toc.html file you saved in the last set of steps.)

8. Find the topic links for each chapter within the file. For example, the following code produces three topic links for Chapter 1:

```
<h2>
<a href="chapter1.html" name="chapter1">Chapter 1</a>
</h2>
   <ul>
      <li>Topic One</li>
      <li>Topic Two</li>
      <li>Topic Three</li>
   </ul>
```

9. Add <a> elements for each topic. The href value is the filename combined with the fragment name. The following code shows the <a> element for the Chapter 1, Topic 1 link.

```
<h2>
<a href="chapter1.html" name="chapter1">Chapter 1</a>
</h2>
   <ul>
      <li><a href="chapter1.html#topic1">Topic One</a></li>
      <li>Topic Two</li>
      <li>Topic Three</li>
</ul>
```

10. Continue to add similar links for each topic listed in the table of contents. When you are finished, your chapter navigation section should look like the following example. The link code is shown in color. (The location of your line breaks might differ.)

```
<h2>
<a href="chapter1.html" name="chapter1">Chapter 1</a>
</h2>
   <ul>
      <li><a href="chapter1.html#topic1">Topic One</a></li>
      <li><a href="chapter1.html#topic2">Topic Two</a></li>
      <li><a href="chapter1.html#topic3">Topic Three</a>
         </li>
   </ul>
<h2>
<a href="chapter2.html" name="chapter2">Chapter 2</a>
</h2>
   <ul>
      <li><a href="chapter2.html#topic1">Topic One</a></li>
      <li><a href="chapter2.html#topic2">Topic Two</a></li>
      <li><a href="chapter2.html#topic3">Topic Three</a>
         </li>
   </ul>
<h2>
<a href="chapter3.html" name="chapter3">Chapter 3</a>
</h2>
   <ul>
      <li><a href="chapter3.html#topic1">Topic One</a></li>
```

```
        <li><a href="chapter3.html#topic2">Topic Two</a></li>
        <li><a href="chapter3.html#topic3">Topic Three</a>
            </li>
    </ul>
```

11. Save the **toc.html** file in the Chapter09 folder of your work folder, and view it in your browser. Test the topic links by selecting a chapter topic and making sure the browser window opens to the correct place in the correct file. Figure 9-10 shows the result of selecting the Chapter 2, Topic 2 link.

    When users click the topic links in the table of contents, the browser opens the destination file and displays the fragment.

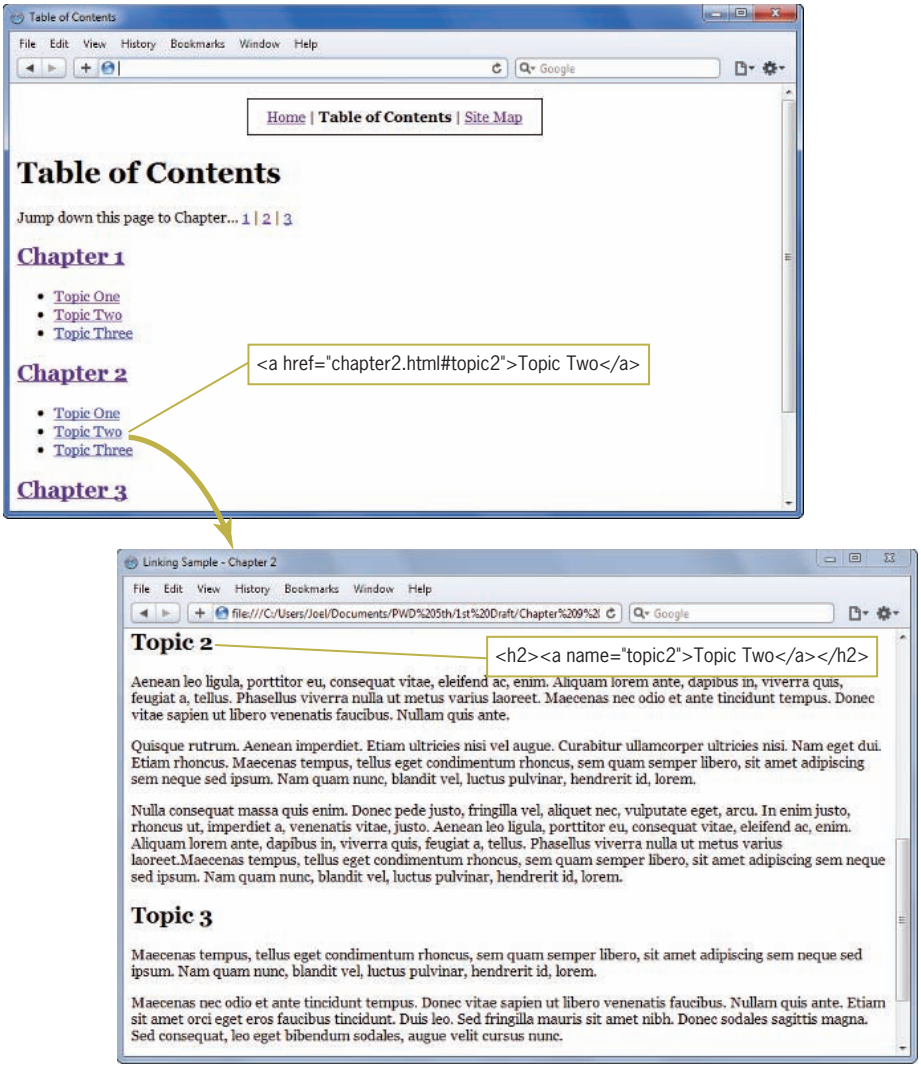


Figure 9-10 Linking to an external fragment

Adding Page Turners

Each chapter file currently contains a navigation bar and fragment identifiers for each topic within the chapter. In this page collection, the user can jump to any file and topic within a file, though some users may want to read the pages sequentially. You can offer this function by adding page-turner links. Page turners let you move either to the previous or next page or section in the collection. These work well in a linear structure of pages, for computer-based learning, or where users read pages or sections in order as shown in Figure 9-11.

Note that Chapter 1 uses the table of contents as the previous page, while Chapter 3 uses the site map as the next page.

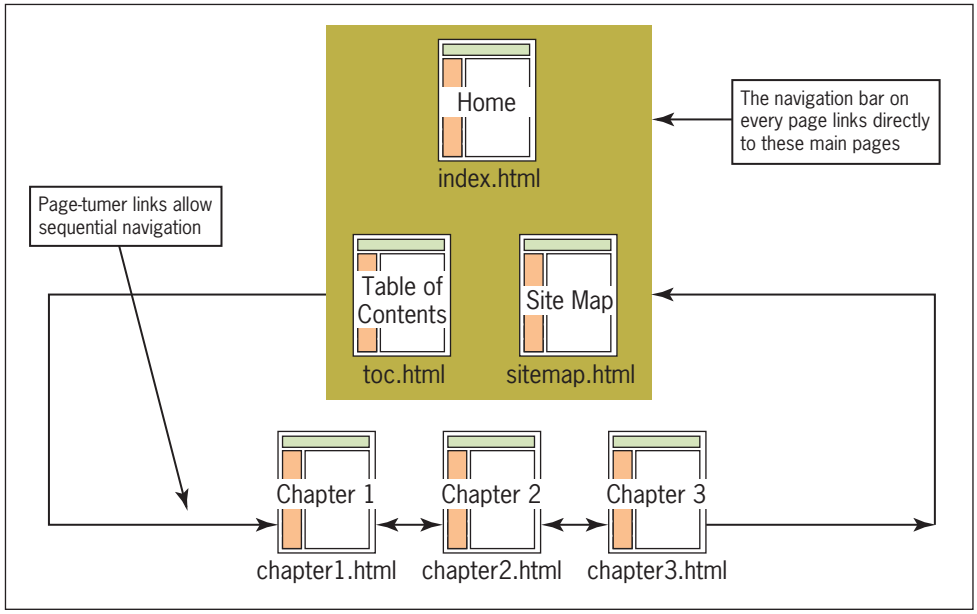


Figure 9-11 Sequential page turning

To add page-turner links:

1. In your editor, open the file **chapter1.html** from the Chapter09 folder of your work folder.
2. Add a <p> element to place the page turner links at the bottom of the page. Set the id attribute to *footernav* as shown in the following code. Insert the code immediately before the closing </body> tag.

```
<p id="footernav">Previous | Chapter 1 | Next</p>
</body>
```

3. Add an <a> element for the previous page. Because this is Chapter 1, make the previous page destination toc.html, as shown in the following code.

```
<p id="footernav"><a href="toc.html">Previous</a> |
   Chapter 1 | Next</p>
```

4. Add an <a> element for the next page. Because this is Chapter 1, make the next page destination chapter2.html, as shown in the following code.

```
<p id="footernav"><a href="toc.html">Previous</a> |
   Chapter 1 |
<a href="chapter2.html">Next</a></p>
```

5. Add a span element with a class attribute to contain the Chapter 1 text. Because this is the Chapter 1 page, the text "Chapter 1" is not a hypertext link but is bold to designate the user's location. You will reuse the *current* style you used previously when building the header navigation bar to indicate the current page. The code looks like this:

```
<p id="footernav"><a href="toc.html">Previous</a> |
<span class="current">Chapter 1</span> |
<a href="chapter2.html">Next</a></p>
```

6. Save the **chapter1.html** file in the Chapter09 folder of your work folder.

7. Now that the HTML code is complete, you will need to add the *footernav* style to the CSS style sheet. Open the stylesheet file **styles.css** in your editor. Locate the headernav style:

```
/* Navigation Header */
#headernav {
   margin-left: auto;
   margin-right: auto;
   padding: 10px;
   border: solid 1px black;
   width: 300px;
   text-align: center;
}
```

8. You can easily match the style of the header navigation bar for the footer by simply adding the *footernav* id selector to the existing style rule as shown:

```
/* Navigation Header */
#headernav, #footernav {
   margin-left: auto;
   margin-right: auto;
   padding: 10px;
   border: solid 1px black;
   width: 300px;
   text-align: center;
}
```

9. Save and close the **styles.css** file. Save the **chapter1.html** file, and then view it in your browser. Your file should now look like Figure 9-12. Test the page-turner links to make sure they point to the correct files.

10. Add the page-turner links to chapter2.html and chapter3.html, changing the Previous and Next links to point to the correct files. Test all your links to make sure they work properly. When you are finished, save and close all the .html files in the Chapter09 folder of your work folder.

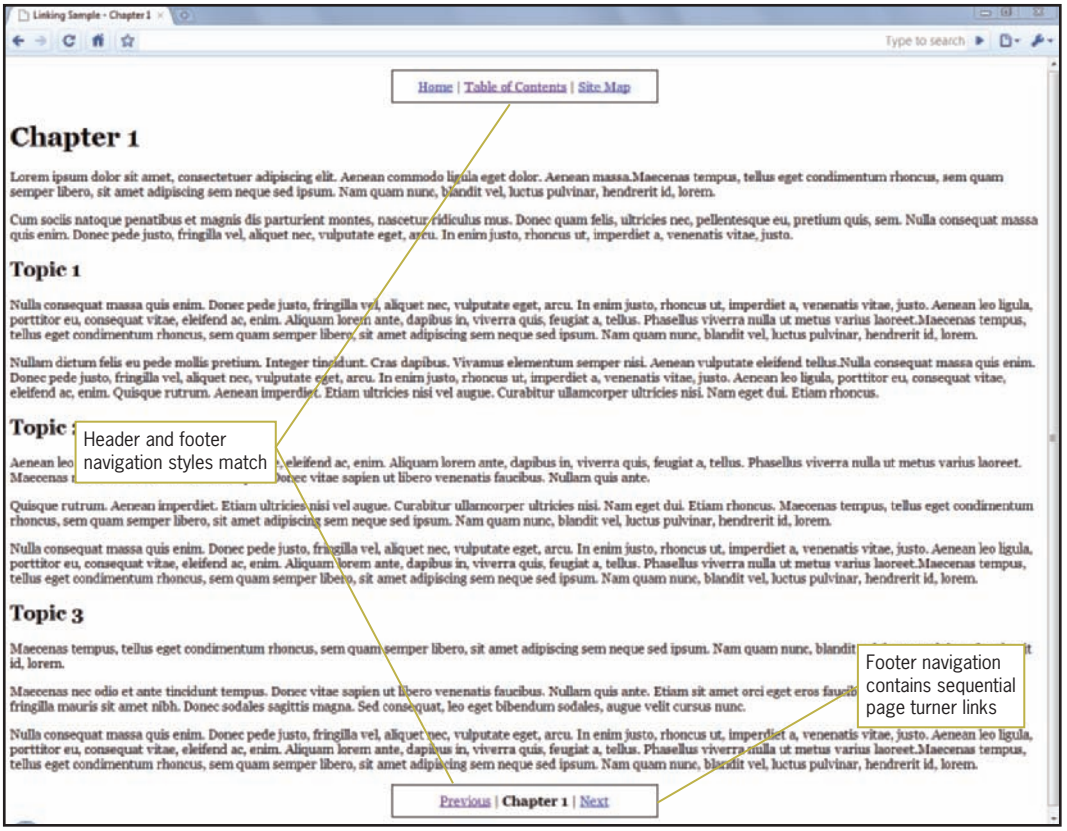


Figure 9-12 Adding page turners in the navigation bar

Adding Contextual Linking

One of the most powerful hypertext capabilities is contextual linking. **Contextual links** allow users to jump to related ideas or cross-references by clicking the word or item that interests them. These are standard hypertext links that you can embed directly in the flow of your content by choosing the key terms and concepts

you anticipate your users will want to follow. Figure 9-13 shows a page from the Wikipedia Web site (*www.wikipedia.com*) that contains contextual linking.

Note the links within the lines of text, which let the user view related information in context. Including the link within a line of text is more effective than including a list of keywords, because users can see related information within the context of the sentence they are reading. Users also can see that repeated words are linked no matter how many times they appear within the browser window, offering users the opportunity to access additional information at any time.

Figure 9-13 Contextual linking

Navigation Summary

You can choose from a variety of navigation options to link a collection of pages. The sample Web pages in this section demonstrate the following text-based linking actions:

- To main pages (Home, Table of Contents, Index)
- To the top of each chapter

- Within the Table of Contents page to chapter descriptions
- From the Table of Contents page to specific topics within each chapter
- Between previous and next chapters
- To related information by using contextual links

Use as many of these options as necessary, but remember to view your content from the user's perspective. Use enough navigation options to allow easy and clear access to your content.

Using Graphics for Navigation and Linking

Although the current Web design trends are towards text-based navigation, there are still many instances where creating a clickable graphic for a link is desirable. To make sure your navigation graphics help rather than hinder your users, use the same graphics consistently throughout your Web site, for the following reasons:

- *To provide predictable navigation cues for the user*—After users learn where to find navigation icons and how to use them, they expect them on every page. Consistent placement and design also build users' trust and help them feel confident that they can find the information they want.
- *To minimize download time*—After the graphic is downloaded, the browser retrieves it from the cache for subsequent pages rather than downloading it every time it appears.

Remember that linked graphics are the result of placing an <img> element within a set of <a> tags. For example:

```
<a href="index.html"><img src="home.gif" alt="home page
image" width="50" height="20" /></a>
```

Refer to Chapter 8 for more information on working with images.

Using the alt Attribute

As you read earlier, you should provide alternate text-based links in addition to graphical links. You can do so by including an alt attribute in the <img> tag of the HTML code for the graphic. Repeating navigation options ensures that you meet the needs of a wide range of users. Some sites choose not to offer a text-based alternative, and this makes it difficult for users who cannot view graphics in their browsers. Figure 9-14 shows the navigation bar from the Barnes and Noble Web site with images turned on and off. Notice that even though images do not appear, the user

can still navigate the site because the <img> elements contain meaningful alt attribute values.

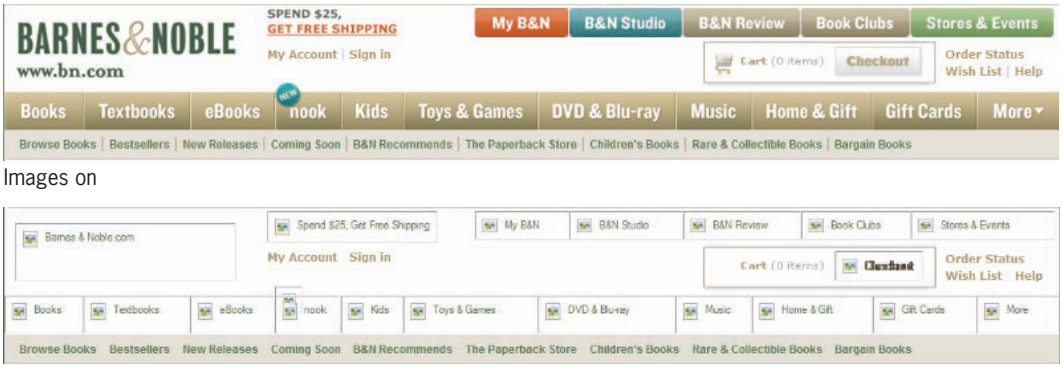


Figure 9-14 Barnes and Noble Web site navigation bar with images on and off

The user finds navigation cues by reading the alt text and pointing to the image areas to find the clickable spots. Accessibility devices can use the alt attribute to provide navigation information as spoken content or in other media. The inclusion of alt attributes is of prime importance to the accessibility of your Web site.

Remember that the alt attribute is different from the title attribute. Alt is designed to provide the alternate text as shown above in Figure 9-14. The title attribute text is displayed in a ToolTip or ScreenTip (a pop-up window that appears when the user points to an object). You can read more about the title attribute in Chapter 8.

Using Icons for Navigation

Figure 9-15 shows the navigation icons from the MapQuest Web site (*www.mapquest.com*). The text labeling on the icons points out one of the main problems with icons—not everyone agrees on their meaning. Especially with a worldwide audience, you never can be sure exactly how your audience will interpret your iconic graphics. This is why so many Web sites choose text-based links, and many that use icons, such as the MapQuest example in Figure 9-15, choose to include descriptive text with the icons.

Figure 9-15 Using icons for navigation

No matter what types of graphics you choose as icons, make sure that your users understand their meaning. Test your navigation graphics on users in your target audience, and ask them to interpret the icons and directional graphics you want to use. The most obvious type of graphics to avoid are symbols that are culturally specific, especially hand gestures (such as thumbs up), which

may be misinterpreted in other cultures. Other graphics, such as directional arrows, or accepted representational graphics, are more likely to be interpreted correctly.

Using Lists for Navigation

Refer to Chapter 5 for more information on list elements.

The HTML list elements are the preferred element for containing navigation links. Lists provide an easy way to create navigation that can be styled with CSS, are accessibility compatible, and work well even if a browser does not support CSS. If for some reason the CSS style rules are not applied, the navigation links will still be shown as a list.

The HTML list elements are commonly used to create bulleted and numbered lists. The following code shows conventional use of the <ul> and <li> elements to create a bulleted list. This list has an id *navlist* that will be used for applying style rules.

```
<ul id="navlist">
   <li><a href="index.html">Home</a></li>
   <li><a href="history.html">History</a></li>
   <li><a href="how.html">How it Works</a></li>
   <li><a href="clubs.html">Balloon Clubs</a></li>
   <li><a href="festivals.html">Festivals</a></li>
   <li><a href="rides.html">Where to Ride</a></li>
   <li><a href="faq.html">FAQ</a></li>
</ul>
```

This code results in the bulleted list shown in Figure 9-16.

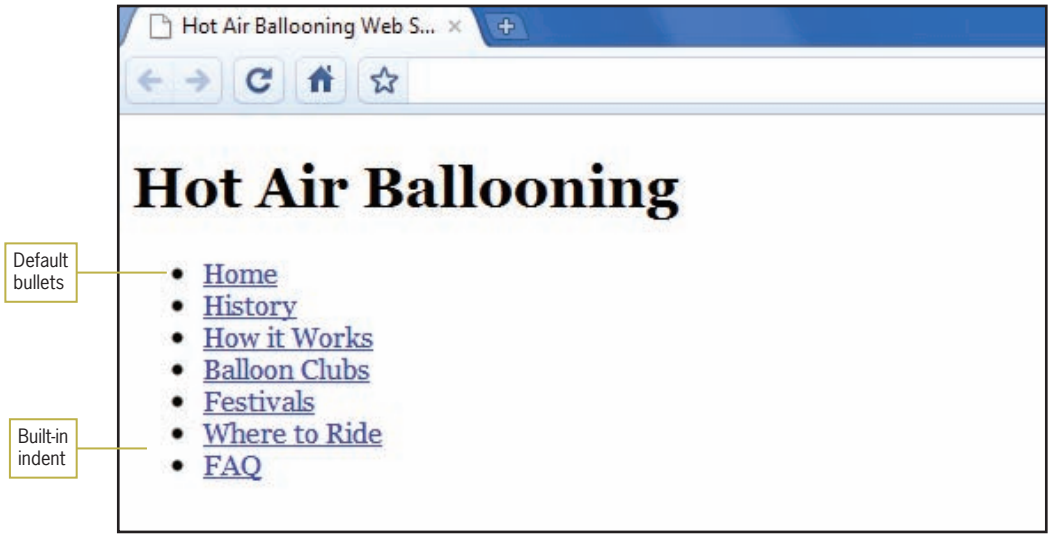


Figure 9-16 Basic unordered list

Removing Default Padding and Margin

The bulleted list in Figure 9-16 has built in spacing that indents the list from the left side of the browser window. Depending on the browser, this built-in spacing is either applied using padding or margin. In most instances you will need to remove this default spacing before creating navigation lists. Set both the margin and padding properties to zero for the <ul> element, as shown in the following code, which selects a <ul> element with an id *navlist*.

```
ul#navlist {
   padding: 0;
   margin: 0;
}
```

Removing Default Bullets

HTML lists come with built-in bullets, which are useful when you are creating standard item lists. When you are building lists for navigation, you can remove the default bullets with the list-style-type property. You can add this to the same style rule you use to remove the default padding and margin, as shown below:

```
ul#navlist {
   padding-left: 0;
   margin-left: 0;
   list-style-type: none;
}
```

The result of removing both the default bullets and indenting results in the list shown in Figure 9-17. This list is ready to be turned into either a horizontal or vertical navigation bar, as you will see in the next sections.

Figure 9-17 List element with default indent and bullets removed

Building Horizontal Navigation Bars

In a standard list element, the list items are block-level elements. A line break separates each item onto its own new line. To create a horizontal navigation bar using a list, you need to set each list item's display setting to *inline*, allowing the list to be displayed without line breaks. You can do this with the display property. The following style rule uses an id selector and descendant selection to select the <li> elements within a <ul> element with an id *navlist*. The style rule sets the display property to *inline*.

Refer to Chapter 6 for more information on the display property and Chapter 4 for more information on CSS selectors.

```
ul#navlist li{
   display: inline;
}
```

The horizontal navigation bar in Figure 9-18 is styled with three styles rules: one that styles the <ul> container, one that sets the <li> elements to inline, and a third that styles the <a> elements that create the hypertext links.

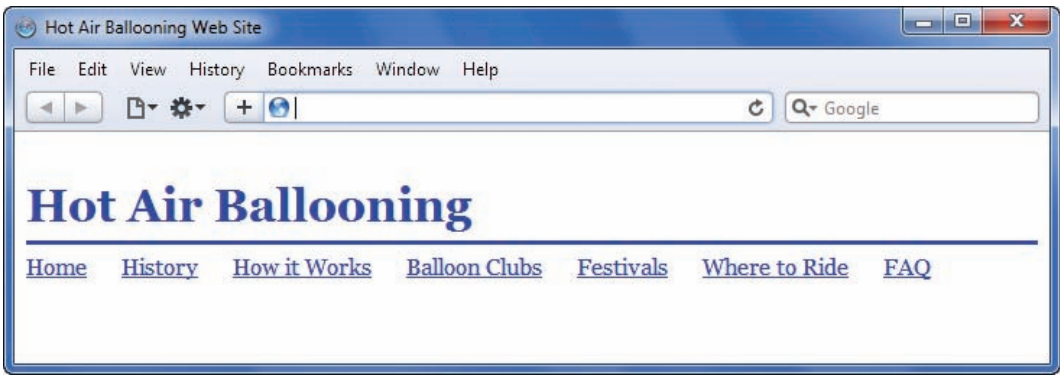


Figure 9-18 Unordered list styled as horizontal navigation

The first style rule removes the default spacing and bullets from the unordered list:

```
ul#navlist {
   padding: 0;
   margin: 0;
   list-style-type: none;
}
```

The second style rule sets the <li> elements to inline display:

```
ul#navlist li{
   display: inline;
}
```

The third style rule styles the <a> elements that contain the link text. In this example, a right margin is added to each link to provide some white space between each link in the horizontal list.

```
ul#navlist a{
   margin-right: 20px;
   }
```

Customizing the Horizontal Navigation Bar

Once you have created the basic list, you can customize it using different CSS style properties. You can remove underlining, add borders and background colors or images, and set space between buttons, all with a few style rules. For example, you can build on the basic horizontal navigation bar in Figure 9-18, which was created with these rules:

```
ul#navlist {
   padding: 0;
   margin: 0;
   list-style-type: none;
}
```

```
ul#navlist li{
   display: inline;
}

ul#navlist a{
   margin-right: 20px;
}
```

Add a border and background color to the <a> element that contains the link text, and set the right margin from 20px to 5px, bringing the link elements closer together.

```
ul#navlist {
   padding: 0;
   margin: 0;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
}

ul#navlist a{
   margin-right: 5px;
   border: solid 1px
   background-color: #ccccff;
}
```

Add 5 pixels of top and bottom padding and 10 pixels of left and right padding. Set the text decoration to *none* to remove the underlining from the links.

```
ul#navlist {
   padding: 0;
   margin: 0;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
}

ul#navlist a{
   margin-right: 5px;
   border: solid 1px red;
   background-color: #ccccff;
   padding: 5px 10px 5px 10px;
   text-decoration: none;}
```

Finally, add a top margin to the <ul> element to offset the entire navigation bar from the heading above.

```
ul#navlist {
   padding: 0;
   margin: 10px 0px 0px 0px;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
}

ul#navlist a{
   margin-right: 5px;
   border: solid 1px;
   background-color: #ccccff;
   padding: 5px 10px 5px 10px;
   text-decoration: none;}
```

This code results in the enhanced navigation bar shown in Figure 9-19.

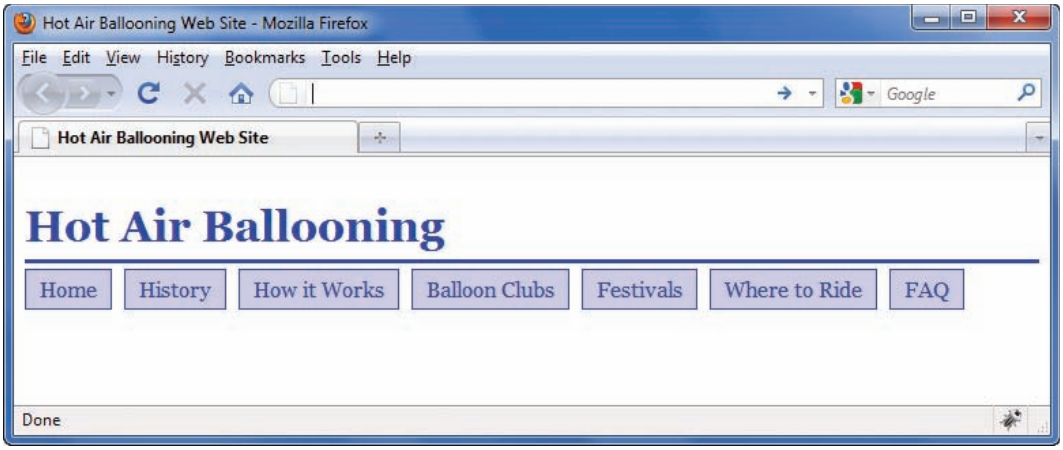


Figure 9-19 Enhanced horizontal navigation bar

Controlling Navigation Bar Width

Your horizontal navigation bar will wrap if a user makes their browser window small enough, as shown in Figure 9-20. To keep this from happening, add a width property to the <ul> element style rule, as shown in the following code:

```
ul#navlist {
   padding: 0;
   margin: 10px 0px 0px 0px;
   list-style-type: none;
   width: 700px;
}
```

Figure 9-20 Fix this undesirable wrapping with the width property

Controlling Navigation Button Width

In the previous example, the horizontal navigation bar buttons are based on the size of the text they contain. You may want to have all navigation buttons be the same width. To create same-size buttons, you will have to change the display type of the <a> elements to *block* to set a consistent width, and then float the boxes so they will align next to each other. The result is the navigation bar shown in Figure 9-21.

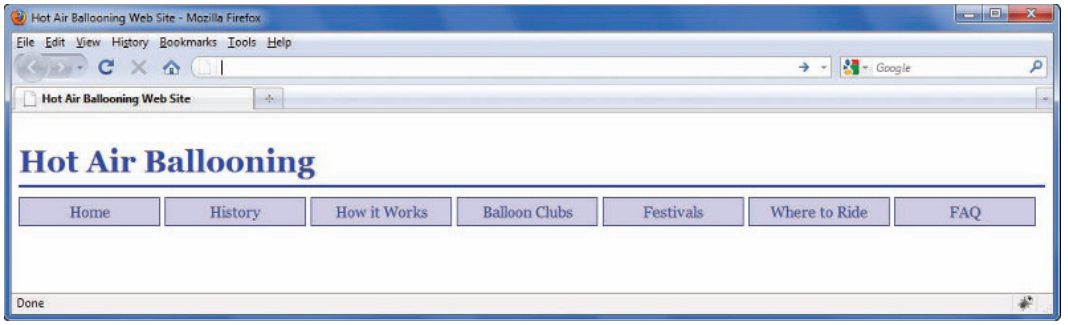


Figure 9-21 Horizontal navigation bar with consistent button width

To accomplish this, add a float property to the <li> elements within the list:

```
ul#navlist {
   padding: 0;
```

```
   margin: 10px 0px 0px 0px;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
   float: left;
}
```

Set properties for the <a> elements that contain the hypertext links. Set the display property to *block* so you can set a width for the buttons. Choose a width that will contain the longest piece of text in your link buttons, in this case 7em. Align the text to center with the text-align property.

```
ul#navlist a{
   margin-right: 5px;
   border: solid 1px red;
   background-color: #ccccff;
   padding: 5px 10px 5px 10px;
   text-decoration: none;
   display: block;
   width: 7em;
   text-align: center;
```

Ems are a good choice for navigation button width because they adapt to the user's font size. If the user chooses a larger or smaller default font for their browser, the navigation buttons resize accordingly.

Building Vertical Navigation Bars

When you are building a vertical navigation bar, you can use a standard list structure without changing the display type as you did for a horizontal navigation bar. The common style of vertical navigation bars usually includes buttons that are the clickable links to different areas of a Web site. These can be styled in an endless variety of ways.

As you saw in the horizontal navigation bar, the <a> elements are contained within <li> elements to create the clickable hypertext links. Because <a> elements are inline by default, you will need to set each <a> element's display setting to *block*. This will let you create clickable buttons of any width you choose.

This example uses the same HTML list elements as you saw in the horizontal example. What is interesting about this example is that you can produce two very different results, in this case both a horizontal and a vertical navigation bar with the same HTML code, just by varying the CSS style rules.

```
<ul id="navlist">
   <li><a href="index.html">Home</a></li>
   <li><a href="history.html">History</a></li>
   <li><a href="how.html">How it Works</a></li>
   <li><a href="clubs.html">Balloon Clubs</a></li>
```

```
   <li><a href="festivals.html">Festivals</a></li>
   <li><a href="rides.html">Where to Ride</a></li>
   <li><a href="faq.html">FAQ</a></li>
</ul>
```

To create a vertical navigation bar, start by setting the margin and padding to zero and removing the default bullet:

```
ul#navlist {
   margin: 0;
   padding: 0;
   list-style-type: none;
}
```

Now style the <a> elements that reside within the <li> elements. Set the display to *block* for the links. Remove the underlining from the link text with the text-decoration property. Set a width of 140 pixels, and add a 2-pixel-wide solid blue (#0033cc) border.

```
ul#navlist li a {
   text-decoration: none;
   display: block;
   width: 140px;
   border: 2px solid #0033cc;
}
```

This style rule results in the incomplete navigation bar in Figure 9-22. You can see the basic structure, but it needs some more style rules to look better.

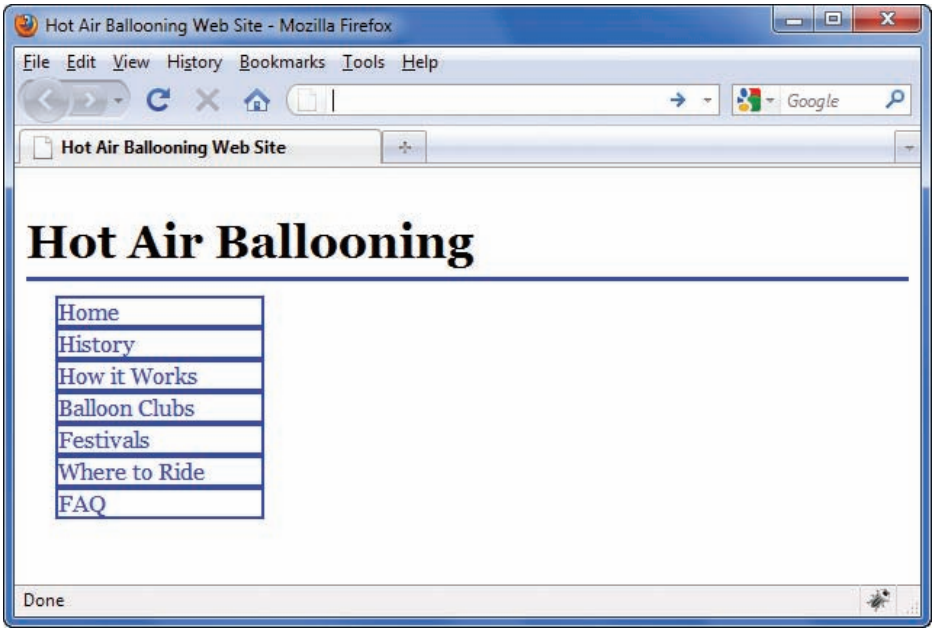


Figure 9-22 Building the vertical navigation bar

To finish the navigation bar, add 5 pixels of padding to offset the link text from the borders of the button. Set the background color to a light blue (#ccccff), and change the text color to black (#000). Finally, add 5 pixels of top margin to separate the buttons vertically.

```
ul#navlist li a {
   text-decoration: none;
   display: block;
   width: 140px;
   border: 2px solid #0033cc;
   padding: 5px;
   background-color: #ccccff;
   color: #000;
   margin-top: 5px;
}
```

Adding this code results in the finished navigation bar shown in Figure 9-23.

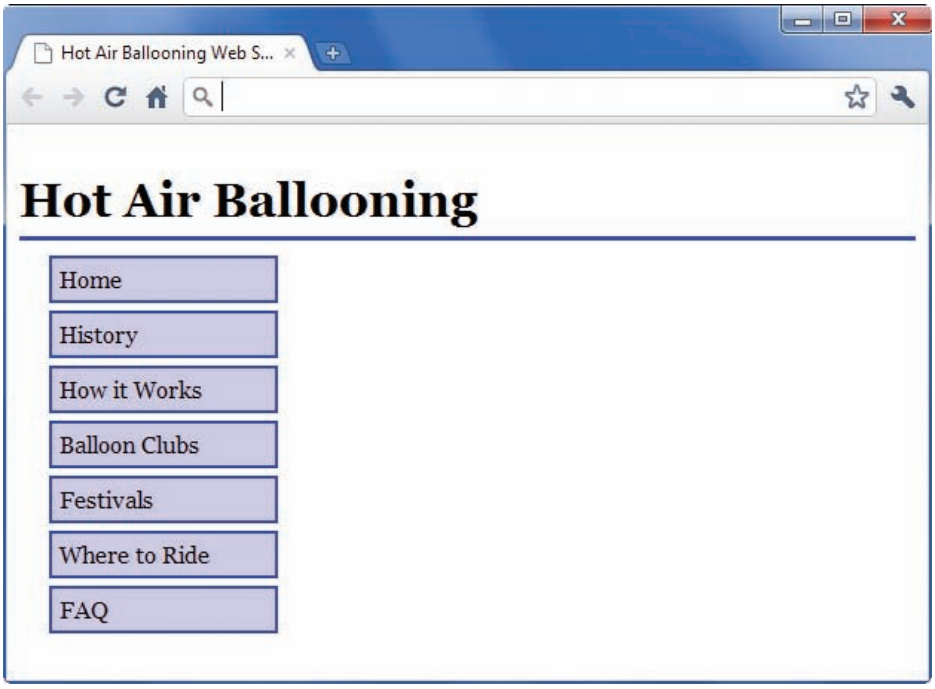


Figure 9-23 Unordered list styled as vertical navigation

Using Background Color and Graphics to Enhance Navigation

Refer to Chapter 8 for more information on using background graphics.

You can use background colors and graphics in a variety of ways to enhance your navigation. You can indicate location with a graphic or by changing a background color. You can create an interactive hover that changes a color or background when the user points to a navigation link.

Indicating History

You can use the link pseudo-classes (described in Chapter 4) along with CSS background images to show a user where they have been on your Web site. Figure 9-24 shows a simple table of contents where the user's visited links are indicated with a red check mark. This keeps track of which pages the user has visited in the site.

Figure 9-24 Indicating history with a background graphic

To create this effect, you must leave enough room in the background area of the element to display the image. In this example, the <a> element content needs to be moved in from the left margin by 30 pixels, which is accomplished by the following style rule. Note that this style rule affects only <a> elements with a *class="chapter"* attribute, so that other <a> elements on the page are not affected.

```
a.chapter {padding-left: 30px;}
```

The result of this rule is that 30 pixels of white space are added to the left of the text, leaving room for the check mark to show, as illustrated in Figure 9-25.

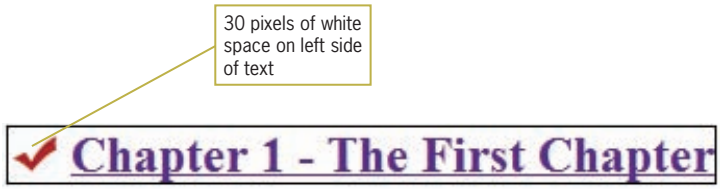


Figure 9-25 Using padding-left to make room for the check mark

The selector in the following style rule applies the background graphic only to <a> elements that have a state of *visited* and a *class="chapter"* attribute.

```
a:visited.chapter { }
```

You can then specify the background graphic location, repetition, and positioning. The background image property points to the location of the file. The background-repeat property specifies that the image should only appear once. Finally, the background-position property states that the image is aligned horizontally to the left and vertically to the center height of the element.

```
a:visited.chapter {
   background-image: url(redcheck.jpg);
   background-repeat: no-repeat;
   background-position: left center;
}
```

The result of these styles is that the user will see a check mark next to any visited pages.

Indicating Location

Location can be indicated by a change in text weight, text color, background color, or with an indicating graphic, which is commonly found on the Web. Figure 9-26 shows a horizontal navigation bar from the AIGA Web site with a background color indicating the current page.

Figure 9-26 Navigation bar indicates current location

Indicating location can be as simple as using bold text instead of a link in a navigation bar on a page, as shown in Figure 9-27.

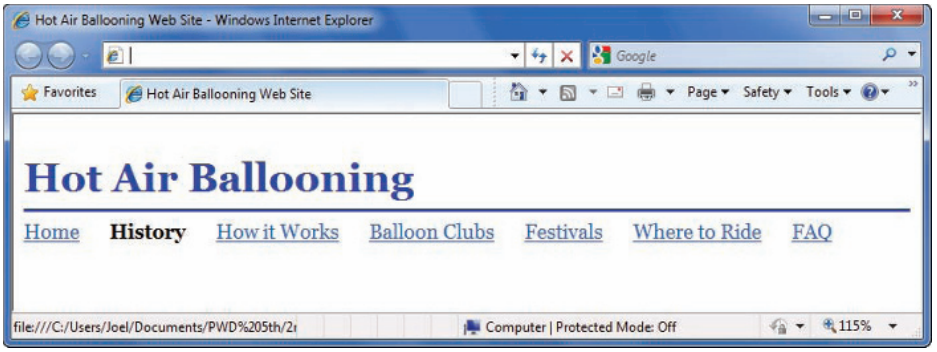


Figure 9-27 Indicating location with bold text

This is easy to accomplish with a style named *current*, for example. This style will be applied to the link text for the current page. The style rule looks like the following:

```
#current {
   font-weight: bold;
}
```

The *current* style rule can then be applied with a element, identifying the current page in the code for the navigation bar as shown:

```
<ul id="navlist">
   <li><a href="home.html">Home</a></li>
   <li><span id="current">History</span></li>
   <li><a href="how.html">How it Works</a></li>
   <li><a href="clubs.html">Balloon Clubs</a></li>
   <li><a href="festivals.html">Festivals</a></li>
   <li><a href="rides.html">Where to Ride</a></li>
   <li><a href="faq.html">FAQ</a></li>

</ul>
```

This same type of style rule can be used to change any number of characteristics for the current page text. For example, to change the background color for the current page text, use the background-color property as shown:

```
#current {
   background-color: #f90000;
}
```

Creating Hover Rollovers

See Chapter 4 for more information on using the :hover pseudo-class.

You can use the CSS :hover pseudo-class with a variety of effects to add interactivity when users scroll over a list of navigation links or buttons. You can change text colors, background colors, and background images based on user actions.

Changing Text Color and Background Color on Hover

Figure 9-28 shows an interactive hover that changes the text color and background color when the user points the mouse pointer at any of the links in the navigation bar.

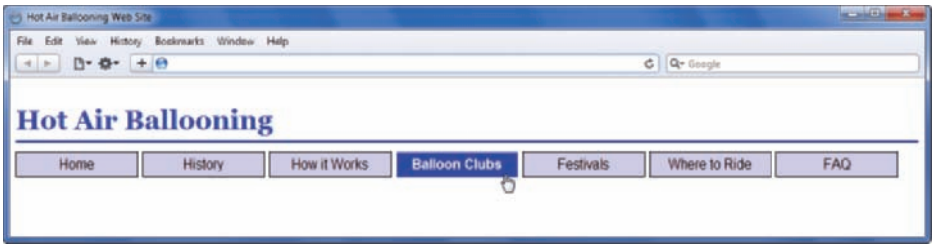


Figure 9-28 Hovering the mouse changes link text and background color

Here are the style rules for the navigation bar. This is the same code you saw before in the horizontal navigation bar in Figure 9-21.

```
ul#navlist {
   padding: 0;
   margin: 10px 0px 0px 0px;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
   float: left;
}

ul#navlist a{
   font-family: arial, sans-serif;
   font-weight: bold;
   margin-right: 5px;
   border: solid 1px;
   padding: 5px 10px 5px 10px;
   text-decoration: none;
   color: #000;
   background-color: #ccccff;
   width: 125px;
   text-align: center;
   display: block;
}
```

The hover effect is created with the following style rule. Note that this selects only <a> elements within the <ul> element that have an *id="navlist"* element; otherwise, it would affect all <a> elements on the page. When the user hovers the mouse over the link, the text color changes to white (#fff), and the background color changes to bright blue (#0033cc). The font-weight property makes the text bold.

```
ul#navlist a:hover {
   color: #fff;
   background-color: #0033cc;
   font-weight: bold;
}
```

Changing Background Images on Hover

You can change background images as easily as changing background colors. For example, Figure 9-29 shows a navigation bar composed of three-dimensional button graphics. When the user hovers the pointer over a button, the button changes color from light blue to dark blue. The buttons are CSS background images that are displayed behind the link text and switch from the light blue button to the dark blue button based on the user action.

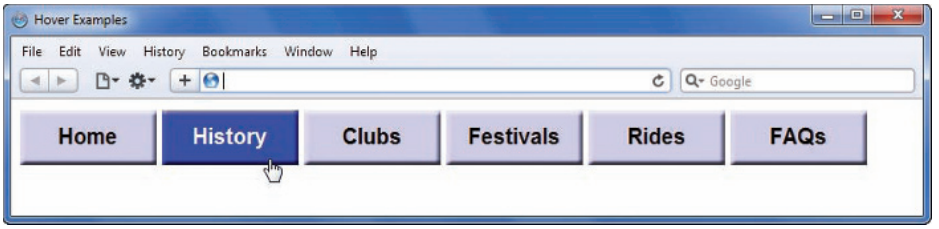


Figure 9-29 Navigation bar with 3-D buttons

Figure 9-30 shows the two buttons used in this navigation bar.

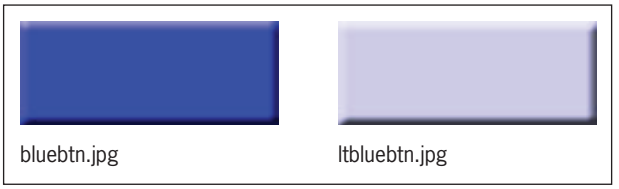


Figure 9-30 Navigation bar background button images

Here are the style rules that make up the entire navigation bar. Like the example earlier, the <a> element style rule includes the

font characteristics for the link text, along with some padding and margin properties. The width and height of the <a> element matches the width and height of the button area, making the entire button clickable. Note the background image specifies ltbluebtn.jpg.

```
ul#navlist {
   padding: 0;
   margin: 10px 0px 0px 0px;
   list-style-type: none;
}

ul#navlist li{
   display: inline;
   float: left;
}

a.navbutton{
   font-family: arial;
   font-size: 1.25em;
   font-weight: bold;
   padding-top: 12px;
   margin-right: 5px;
   text-decoration: none;
   width: 125px;
   height: 50px;
   text-align: center;
   display: block;
   color: #000;
   background-image: url(ltbluebtn.jpg);
   background-repeat: no-repeat;

}
```

The hover effect is created with the following style rule. Note that this selects only <a> elements with a class="navbutton" attribute; otherwise, it would affect all <a> elements on the page. When the user hovers the pointer over the link, the background image changes from ltbluebtn.jpg to bluebtn.jpg.

```
a.navbutton:hover {
   background-image: url(bluebtn.jpg)
   color: #fff;
}
```

Underlining on Hover

Many Web sites disable the default underlining of hypertext links. The CSS text-decoration property sets the value to *none*, which turns off hypertext linking as shown in the following code:

```
a {text-decoration: none;}
```

You can also use background colors to highlight links using the background-color property, as described in the Chapter 4 section titled "Using the :hover Pseudo-Class."

You can use the hover pseudo-class to turn the underlining on when the user points to the link. The style rule looks like this:

```
a:hover {text-decoration: underline;}
```

You can also create other linking effects by substituting a bottom border instead of the standard text underlining. In this style rule, the border-bottom property is used to display a dashed blue line when the user points at the link, as shown in Figure 9-31.

```
a:hover {border-bottom: dashed blue;}
```

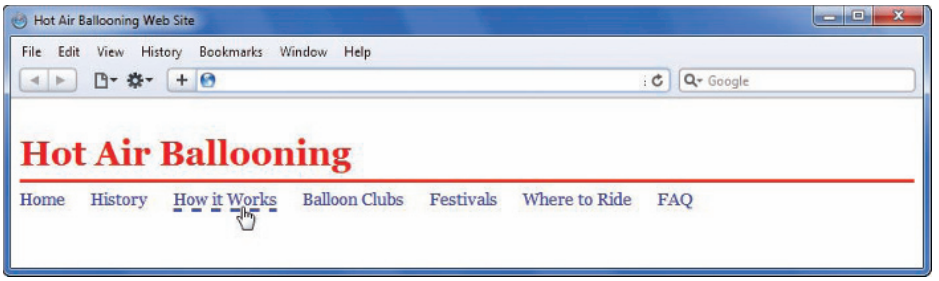


Figure 9-31 Underlining a link with the :hover pseudo-class

Chapter Summary

Usable navigation is the result of working with the power of hypertext and designing for your users' needs. Keep the following points in mind:

- Work from the user's point of view. Think about where users want to go within your Web site, and make it easy for them to get there.
- Add plenty of links to make all areas of your Web site quickly accessible to your users. Link to fragments as well as whole pages. Make it easy to get back to your navigation options.
- In addition to providing links, make sure you provide plenty of location cues to let users know where they are.
- Use text-based navigation bars to link users to other pages in your site. Use other text-based links to help users move through a long page of information or through a table of contents.
- Consider text as an alternative to graphical links. Every graphic adds to download time. When using graphics and icons as

navigational links, make sure users can interpret these links correctly by including text as part of the images. Also, be sure to use navigation icons consistently throughout your Web site to provide predictable cues for users and to minimize download time.

- Include alt values to your <img> tags to provide alternate navigation options for users.
- Use CSS to build attractive horizontal and vertical navigation bars using simple list elements.
- You can use background colors, text colors, and graphics to enhance navigation by indicating history or location.
- You can use the hover pseudo-class to add interactivity to navigation.

Key Terms

breadcrumb path—A series of links, usually at the top of a Web page, that shows the user's location within the site hierarchy. Users can click any of the links in the path to move through the content structure.

contextual link—A link that allows users to jump to related ideas or cross-references by clicking the word or item that interests them. You can embed contextual links directly in your content by choosing the key terms and concepts you anticipate your users will want to follow.

fragment identifier—The use of the <a> element and name attribute to name a segment of an HTML file. You then can reference the fragment name in a hypertext link.

Review Questions

1. List three advantages of linking by using text instead of graphics.

2. What four navigation questions should the user be able to answer?

3. List three types of navigation cues.

4. List three ways to control information overload when designing a Web site.

5. Explain why you would include both graphic and text-based links on a Web page.

6. List two navigation cues you can add to a text-based navigation bar.

7. Why is it best to make <a> the innermost element to a piece of text?

8. What <a> tag attribute is associated with fragment identifiers?

9. List two ways to break up lengthy HTML pages.

10. What character entity is useful as an invisible link destination?

11. What attribute do you use to make an <a> tag both a source and destination anchor?

12. How do you link to a fragment in an external file?

13. Page turners work best in what type of structure?

14. What are the benefits of contextual linking?

15. List two reasons for standardizing graphics.

16. What are the benefits of using navigation graphics?

17. What are the drawbacks of using navigation icons?

18. What are the benefits of using the alt attribute?

Hands-On Projects

1. This book's Online Companion Web site contains all the HTML files for the sample Web site illustrated in Figure 9-2. Use these sample HTML files to build an alternate navigation scheme. Refer to the information structure illustrations in Chapter 3 (Figure 3-8 through

Figure 3-16) for examples of different navigation models. Choose a structure and code examples of usable navigation for the model.

2. In this project, you build a horizontal navigation bar. The code you add to the file in the following steps appears in blue.

   a. Copy the **horizontal nav bar.html** file from the Chapter 09 folder provided with your Data Files to the Chapter09 folder in your work folder. (Create the Chapter 09 folder, if necessary.)

   b. Start your text editor, and open the file **horizontal nav bar.html**. The HTML code is an unordered list with an id of *navlist.*

```
<ul id="navlist">
<li><a href=" ">Welcome</a></li>
<li><a href=" ">Services</a></li>
<li><a href=" ">Portfolio</a></li>
<li><a href=" ">About Us</a></li>
<li><a href=" ">Contact Us</a></li>
<li><a href=" ">FAQ</a></li>
</ul>
```

   c. Open the **horizontal nav bar.html** file in a browser. The list should look like Figure 9-32.

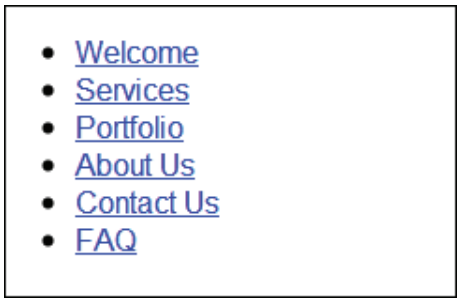


Figure 9-32 Unordered list in the beginning horizontal nav bar file

   d. In the style section, create a selector for the navigation list:

```
ul#navlist { }
```

e. Add style properties to remove the default spacing and bullets from the unordered list:

```
ul#navlist {
   padding: 0;
   margin: 0;
   list-style-type: none;
}
```

f. Write another rule that selects the <li> elements and sets them to inline display:

```
ul#navlist li{
   display: inline;
}
```

g. Write a third style that selects the <a> elements containing the link text. Add a right margin to provide some white space between each link in the horizontal list:

```
ul#navlist a{
   margin-right: 20px;
   }
```

h. Save your file and view it in your browser. The finished navigation bar looks like Figure 9-33.

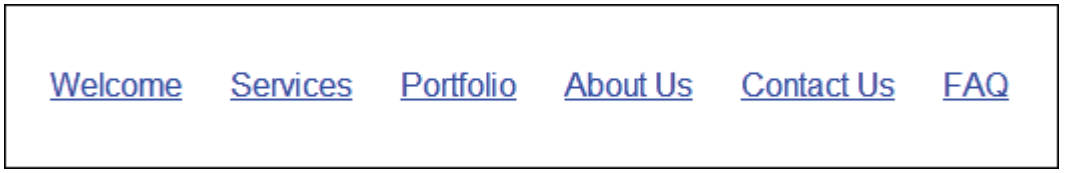


Figure 9-33 Completed horizontal nav bar

3. In this project, you build a vertical navigation bar. The code you add to the file in the following steps appears in blue.

a. Copy the **vertical nav bar.html** file from the Chapter 09 folder provided with your Data Files to the Chapter09 folder in your work folder. (Create the Chapter09 folder, if necessary.)

b. Start your text editor, and open the file **vertical nav bar.html**. The HTML code is an unordered list with an id of *navlist.*

```
<ul id="navlist">
<li><a href=" ">Welcome</a></li>
<li><a href=" ">Services</a></li>
```

```
<li><a href=" ">Portfolio</a></li>
<li><a href=" ">About Us</a></li>
<li><a href=" ">Contact Us</a></li>
<li><a href=" ">FAQ</a></li>
</ul>
```

c. Open the **vertical nav bar.html** file in a browser. The list should look like Figure 9-34.

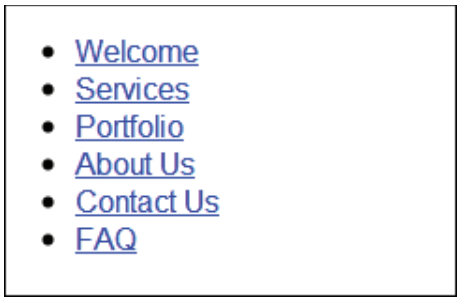


Figure 9-34 Unordered list in the beginning vertical nav bar file

d. In the style section, create a selector for the navigation list:

```
ul#navlist { }
```

e. Add style properties to remove the default spacing and bullets from the unordered list:

```
ul#navlist {
   padding: 0;
   margin-left: 30px
   list-style-type: none;
}
```

f. Create a second rule that selects the <a> elements residing within the <li> elements:

```
ul#navlist li a { }
```

g. Add style rules that set the display to *block* for the links. Remove the underlining from the link text with the text-decoration property. Set a width of 140 pixels, and add a 2-pixel solid border.

```
ul#navlist li a {
   text-decoration: none;
   display: block;
   width: 140px;
   border: 2px solid;
}
```

h. Finish the navigation bar by adding 5 pixels of padding to offset the link text from the borders of the button. Also add 5 pixels of top margin to separate the buttons vertically.

```
ul#navlist li a {
   text-decoration: none;
   display: block;
   width: 140px;
   border: 2px solid;
   padding: 5px;
   margin-top: 5px;
}
```

i. Save your file and view it in your browser. The finished navigation bar looks like Figure 9-35.

Figure 9-35 Completed vertical nav bar

4. Browse the Web and find a Web site that has a successful navigation design. Write a short summary of why the navigation is effective and how it fits the user's needs. Consider the following criteria:

   a. Are the linking text and images meaningful?

   b. Is it easy to access the site's features?

   c. Can you easily search for content?

   d. Is there a site map or other site-orienting feature?

5. Find an online shopping Web site.
    a. Examine the navigation options, and indicate whether you think the navigation adds to or detracts from the online shopping experience.
    b. Describe how to change the navigation to increase its effectiveness.

6. Find an online information resource likely to be used for research. Examine the navigation options, and describe how the navigation helps or hinders the user's information-searching process. Consider the following:
    a. How cluttered is the user interface? Does it deter finding information?
    b. Is navigation prominent or secondary to the page design? Does the user always know his or her location in the site? Is the linking text concise and easy to understand? Is the link destination easy to determine from the linking text?
    c. How deep is the structure of the site? How many clicks does it take to get to the desired information?

7. Browse the Web to find examples of Web sites that need better navigation options. Using examples from the Web site, describe how you would improve the navigation choices.

8. Browse the Web to find a Web site that uses more than one navigation method, and describe whether this benefits the Web site and why.

9. Find a site that illustrates a navigation method different from the ones described in this chapter. Describe the navigation method and state whether this benefits the Web site and why.

Individual Case Project

Examine the flowchart you created for your Web site. Consider the requirements of both internal and external navigation. Create a revised flowchart that shows the variety of navigation options you are planning for the Web site.

Using your HTML editor, mark up examples of navigation bars for your content. Make sure your filenames are intact before you start coding. Save the various navigation bars as separate HTML files for later inclusion in your Web pages.

Plan the types of navigation graphics you want to create. Sketch page banners, navigation buttons, and related graphics. Find sources for navigation graphics. For example, you can use public domain (noncopyrighted) clip art collections on the Web for basic navigation arrows and other graphics.

Team Case Project

Work as a team to examine and discuss the flowchart you created for your Web site. Consider the requirements of both internal and external navigation. Collaborating as a team, refine your work to create a revised flowchart that shows the variety of navigation options you are planning for the Web site. Make sure you have standardized your filenames and all team members agree on the naming conventions.

Work individually to create text-based navigation bars for your content. Use your HTML editor to mark up examples of navigation bars for site navigation. Save the various navigation bars as separate HTML files. Have a team meeting where all members can present their navigation schemes. Choose the best scheme or combine features from different schemes to create the navigation for the group's Web site.

Work together to plan the types of navigation graphics you want to create. Assign team members to complete the remaining tasks, which include sketching page banners, navigation buttons, and related graphics, as well as finding sources for navigation graphics. For example, you can use public domain (noncopyrighted) clip art collections on the Web for basic navigation arrows and other graphics.

CHAPTER 10

Data Tables

When you complete this chapter, you will be able to:

- Use table elements
- Use table headers and footers
- Group columns
- Style table borders
- Apply padding, margins, and floats to tables
- Style table background colors
- Apply table styles

The 3.2 release of HTML in 1997 included table elements for the purpose of organizing tabular data in rows and columns. Web designers quickly realized they could use the table elements to build print-like design structures that allowed them to break away from the left-alignment constraints of basic HTML. With tables, Web designers had the control and the tools to build columnar layouts, align text, add white space, and structure pages. This misuse of the table elements, although well intentioned, created problems with Web site accessibility and compatibility. CSS has long offered the potential for an alternate page layout system, but browser support had to catch up before this was a viable method. The flexibility, accessibility, and ease of maintaining layouts created with CSS makes them the clear choice for designing Web pages, as detailed in Chapter 7. Now that CSS page layouts are broadly supported, tables should only be used only to present data as described in this chapter.

Using Table Elements

To build effective tables, you must be familiar with the HTML table elements. This section describes the most commonly used table elements.

HTML tables are designed not only to present data properly in the browser window, but to be read sequentially by screen readers and other assistive devices. Some of the table features available to Web designers include the ability to span rows and columns and to create table header cells that declare the contents of a column of data.

The HTML table elements allow the arrangement of data into rows, cells, and columns. Table 10-1 lists the table elements and their usage.

Element	Description
table	Establishes the table; contains all other elements that specify caption, rows, and content
tr	Table row; contains the table cells
td	Table data cell; contains the table data
th	Table header cell; contains header information for a column of data
caption	Provides a short description of the table's contents
thead	Signifies table header

Table 10-1 HTML table elements *(continues)*

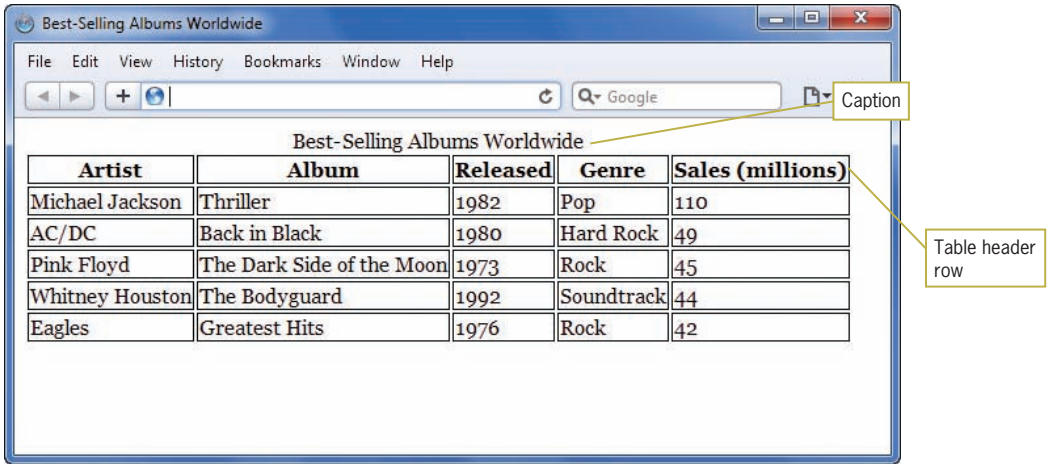
(continued)

tbody	Signifies table body
tfoot	Signifies table footer
col	Specifies column properties
colgroup	Specifies multiple column properties

Table 10-1 HTML table elements

The HTML **<table> element** contains the table information, which consists of **table header elements (<th>)**, **table row elements (<tr>)**, and individual **table data cells (<td>)**. These are the three elements used most frequently when you are building tables. Figure 10-1 shows a basic HTML table with a caption, headers, and rows of data. This table also has an accompanying style that sets a border for each of the cells. The style rule looks like this:

```
th, td {border: solid 1px black;}
```



Best-Selling Albums Worldwide

Artist	Album	Released	Genre	Sales (millions)
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Figure 10-1 Basic HTML table

The basic table is the result of the following code:

```
<table>
<caption>Best-Selling Albums Worldwide</caption>
   <tr>
      <th>Artist</th>
      <th>Album</th>
      <th>Released</th>
      <th>Genre</th>
      <th>Sales (millions)</th>
   </tr>
   <tr>
      <td>Michael Jackson</td>
```

```
            <td>Thriller</td>
            <td>1982</td>
            <td>Pop</td>
            <td>110</td>
        </tr>
        <tr>
            <td>AC/DC</td>
            <td>Back in Black</td>
            <td>1980</td>
            <td>Hard Rock</td>
            <td>49</td>
        </tr>
        <tr>
            <td>Pink Floyd</td>
            <td>The Dark Side of the Moon</td>
            <td>1973</td>
            <td>Rock</td>
            <td>45</td>
        </tr>
        <tr>
            <td>Whitney Houston</td>
            <td>The Bodyguard</td>
            <td>1992</td>
            <td>Soundtrack</td>
            <td>44</td>
        </tr>
        <tr>
            <td>Eagles</td>
            <td>Greatest Hits</td>
            <td>1976</td>
            <td>Rock</td>
            <td>42</td>
        </tr>
</table>
```

Table code can get complicated when you add content to your tables. One small error in your code can cause unpredictable results in the browser.

You can simplify your table creation and maintenance tasks by writing clean, commented code. If you use plenty of white space in the code, you will find your tables easier to access and change. Adding comments helps you quickly find the code you want.

The <table> element contains the rows and cells that make up the table. The <tr> tag marks the beginning and end of each of the five rows of the table. Notice that the <tr> tag contains the table cells, but no content of its own.

You may occasionally use the <caption> and <th> elements when creating tables. The **<caption> element** lets you add a caption to the table. By default, captions are displayed at the top of the table. The <caption> element must appear immediately after the opening table tag, as shown in the code sample for Figure 10-1. You will see how to use CSS to style the caption element later in this chapter.

The <th> element lets you create a table header cell that presents the cell content as bold and centered by default.

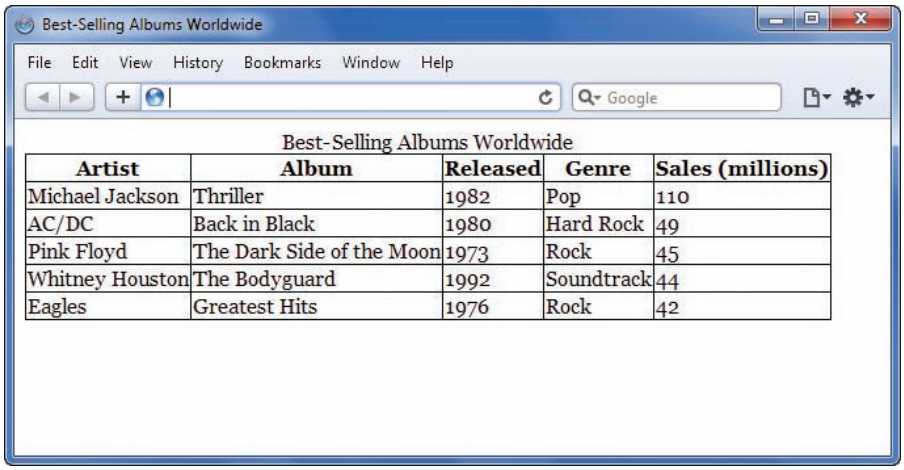
Collapsing Table Borders

Notice that each cell in the table shown in Figure 10-1 has its own border. This can make the data in the table difficult to read. This table will be more legible with the table borders collapsed. You can specify this with the CSS border-collapse property.

border-collapse property description	
Value:	separate | collapse
Initial:	separate
Applies to:	<table> element
Inherited:	yes

The following style rule collapses the borders for the table as shown in Figure 10-2.

```
table {border-collapse: collapse;}
```



Best-Selling Albums Worldwide

Artist	Album	Released	Genre	Sales (millions)
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Figure 10-2 Table with borders collapsed

Spanning Columns

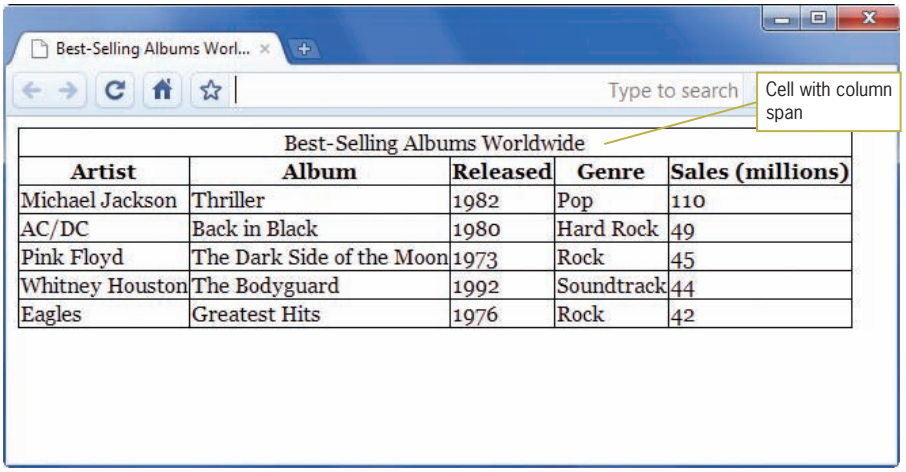
The **colspan** attribute lets you create cells that span multiple columns of a table. Column cells always span to the right. Figure 10-3 shows a table with a column span in the first row.

The following code fragment shows the colspan attribute in blue:

```
<tr>
   <td class="title" colspan="5">
   Best-Selling Albums Worldwide</td>
</tr>
```

Notice that this cell also contains a class="title" attribute. This is used to apply a CSS text-align property to center the text in the cell.

When you build column spans, make sure that all of your columns add up to the correct number of cells. In this code, because each row has five cells, the colspan attribute is set to five to span all columns of the table, as shown in Figure 10-3.



Best-Selling Albums Worldwide				
Artist	**Album**	**Released**	**Genre**	**Sales (millions)**
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Figure 10-3 Table with a column span

Spanning Rows

The **rowspan** attribute lets you create cells that span multiple rows of a table. Rows always span down. Figure 10-4 shows a table with a row span added to the left of the data cells. Adding a row span usually means adding an extra cell to accommodate the span.

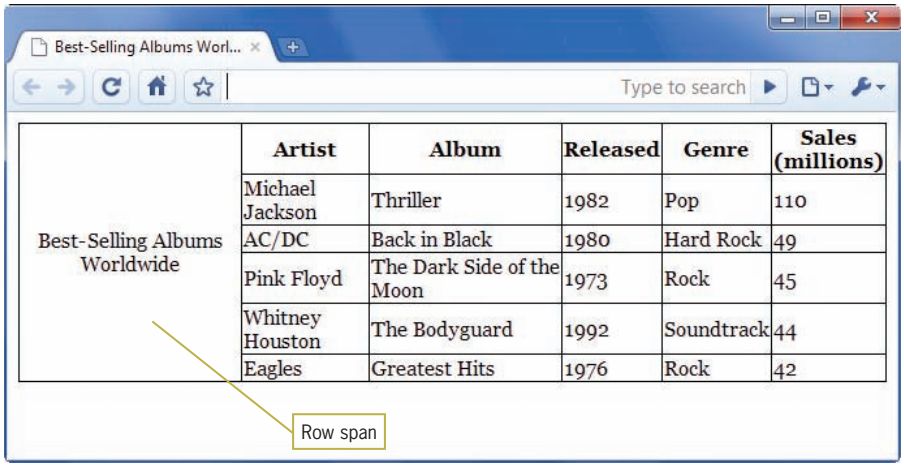


Figure 10-4 Table with row span

The following code shows the new cell that contains the rowspan attribute and the extra column cell in the table header row:

```
<table>
   <tr>
     <td class="title" rowspan="6">
     Best-Selling Albums Worldwide</td>
     <th>Artist</th>
     <th>Album</th>
     <th>Released</th>
     <th>Genre</th>
     <th>Sales (millions)</th>
   </tr>
...rest of the table code...
</table>
```

The row span cell is the first cell in the first row of the table. It spans down through six rows of the table. Often you will see the rowspan value set to "99." This ensures that the rowspan will always be greater than the number of rows in the table (at least for tables that contain less than 99 rows). Once again, note that a class="title" attribute is used to apply a text-align property to the text.

Using Table Headers and Footers

Rows can be grouped into head, body, and footer sections using the <thead>, <tbody>, and <tfoot> elements, as shown in the following code sample.

```
<table>
<thead>
   <tr><th>Header Cell 1</th>
       <th>Header Cell 2</th>
   </tr>
</thead>
<tbody>
   <tr><td>Body Cell 1</td>
       <td>Body Cell 2</td>
   </tr>
   <tr><td>Body Cell 3</td>
       <td>Body Cell 4</td>
   </tr>
</tbody>
<tfoot>
   <tr><td>Footer Cell 1</td>
       <td>Footer Cell 2</td>
   </tr>
</tfoot>
</table>
```

The <thead> and <tfoot> elements can then be styled with style rules. Figure 10-5 shows an example of using CSS style rules to style the header and footer of a table.

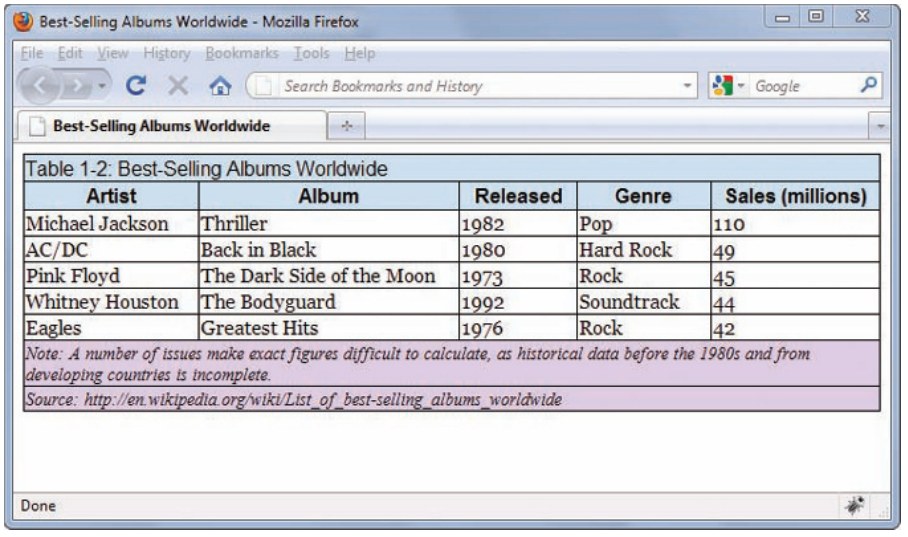


Table 1-2: Best-Selling Albums Worldwide

Artist	Album	Released	Genre	Sales (millions)
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Note: A number of issues make exact figures difficult to calculate, as historical data before the 1980s and from developing countries is incomplete.

Source: http://en.wikipedia.org/wiki/List_of_best-selling_albums_worldwide

Figure 10-5 Table with styled headers and footers

```
thead {
   font-family: arial;
   background-color: #ccddee;
}

tfoot {
   background-color: #ddccee;
   font-family: times, serif;
   font-size: .9em;
   font-style: italic;
}
```

```
<table>
<thead>
   <tr>
   <td colspan="5">Table 1-2: Best-Selling Albums Worldwide
      </caption>
   </tr>
   <tr>
      <th>Artist</th>
      <th>Album</th>
      <th>Released</th>
      <th>Genre</th>
      <th>Sales (millions)</th>
   </tr>
</thead>

...rest of table code...

<tfoot>
   <tr>
   <td colspan="5"> Note: A number of issues make exact figures
difficult to calculate, as historical data before the 1980s and
from developing countries is incomplete.</td>
   </tr>
   <tr>
   <td colspan="5"> Source:
http://en.wikipedia.org/wiki/List_of_best-
selling_albums_worldwide</td>
   </tr>
</tfoot>
</table>
```

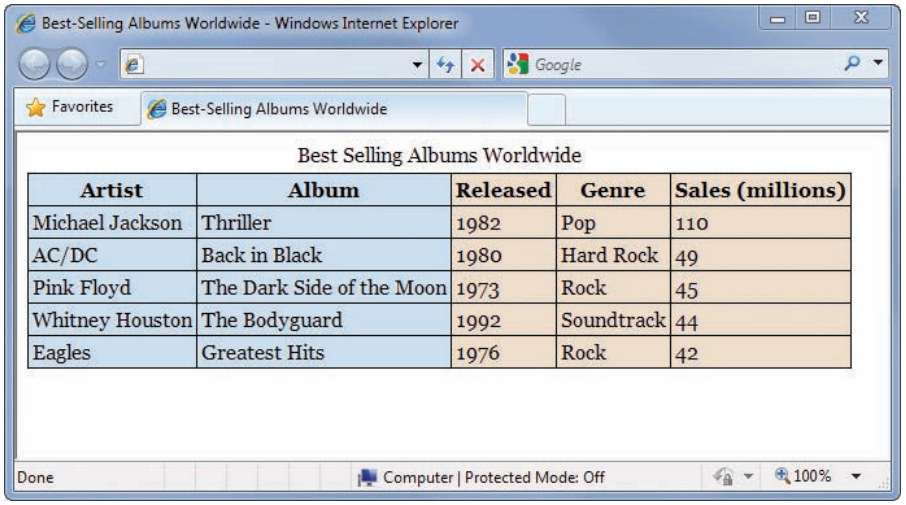
Grouping Columns

The <colgroup> and <col> elements allow you to apply style characteristics to groups of columns or individual columns. The <colgroup> element has a span attribute that lets you set the number of columns specified in the group. Column groups are always applied left to right in the table. The only other available property is width, which lets you specify the width of a column or group of columns.

Only the width property and the background properties, such as background-color and background-image, can be applied to the <colgroup> and <col> elements.

The <col> element lets you specify style characteristics for individual columns. It always appears within a set of <colgroup> tags. Both <colgroup> and <col> elements must appear immediately after the opening <table> element, or after the <caption> element if the table contains a caption.

Figure 10-6 shows the use of the colgroup element applied to the sample table. In this five-column table, the columns are organized into two column groups. The left column group contains two columns, and the right column group contains three columns.



Best Selling Albums Worldwide

Artist	Album	Released	Genre	Sales (millions)
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Figure 10-6 Applying styles to column groups

The code for the table follows. Notice that each column group contains a class name, which is used to apply a CSS background color to the column group. The <colgroup> elements appear directly after the caption element.

```
<table>
<caption>Best Selling Albums Worldwide</caption>
<colgroup span="2" class="left"></colgroup>
<colgroup span="3" class="right"></colgroup>
<tr>
   <th>Artist</th>
   <th>Album</th>
   <th>Released</th>
   <th>Genre</th>
   <th>Sales (millions)</th>
</tr>

...rest of table code...
```

You can use the <col> element to apply class names and style individual columns. The following HTML code in the table shows each column with a class name.

```
<colgroup>
   <col class="artist" />
   <col class="album" />
   <col class="released" />
   <col class="genre" />
   <col class="sales" />
</colgroup>
```

These class names can be used as selectors in the style sheet. The following style rule selects the column with the class name *artist* and applies a background color to the column, as shown in Figure 10-7.

```
col.artist {background-color: #ddeeff;}
```

Figure 10-7 Selecting one column and applying a background color

Styling the Caption

You can style the caption with CSS to position the caption on the top or bottom of the table using the caption-side property. You can also choose from any of the other style properties to enhance the caption text.

caption-side property description	
Value:	top | bottom
Initial:	top
Applies to:	<caption> element
Inherited:	yes

Figure 10-8 shows the caption left-aligned and italic. The style rule that follows shows the use of common CSS properties to specify text alignment, font style, and padding.

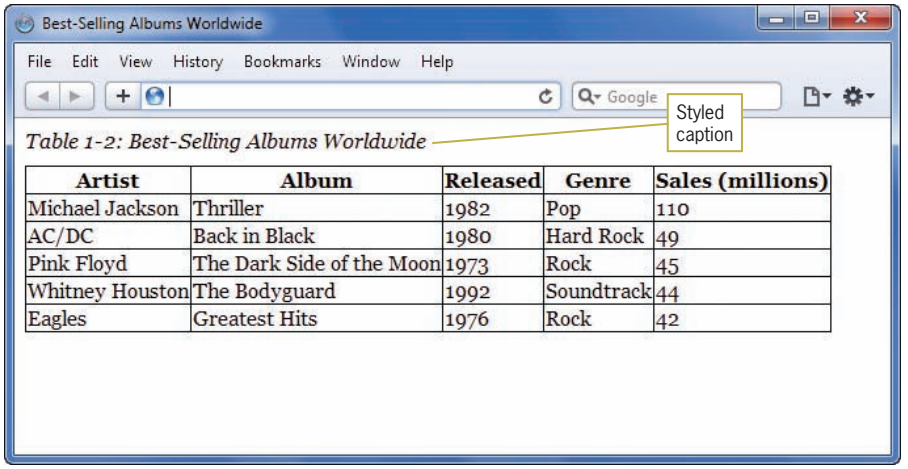


Figure 10-8 Styled caption

```
<style type="text/css">
body {
   font-family: georgia;
}

table {
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
}

caption {text-align: left;
   font-style: italic;
   padding-bottom: 10px;
}
</style>
```

Styling Table Borders

By default, tables are displayed in the browser with borders turned off. You can add borders to tables using CSS style rules. Borders can be applied to the whole table, to individual rows, and to individual cells. Using the table element as a selector

applies the border only to the outside of the table as shown in Figure 10-9.

Figure 10-9 Table with outside border only

The style rule for this table looks like the following code. Note that the border-collapse property is used to remove the extra space between the borders.

```
table {
   border: solid 1px black;
   border-collapse: collapse;
}
```

Recall from Chapter 6 that borders do not inherit styles, so you can add borders for each cell by specifying them in a separate style rule. The sample table has two types of cells: table header cells <th> and table data cells <td>. These must each be added as selectors to make sure every cell has a border. The style rule looks like the following code, and the result is shown in Figure 10-10. Notice that the <th> and <td> elements share the same style declaration.

```
table {
   border: solid 1px black;
   border-collapse: collapse;
}
th, td {
   border: solid 1px black;
}
```

Best-Selling Albums Worldwide

Artist	Album	Released	Genre	Sales (millions)
Michael Jackson	Thriller	1982	Pop	110
AC/DC	Back in Black	1980	Hard Rock	49
Pink Floyd	The Dark Side of the Moon	1973	Rock	45
Whitney Houston	The Bodyguard	1992	Soundtrack	44
Eagles	Greatest Hits	1976	Rock	42

Figure 10-10 Table with outside border and cell borders

You can also style individual row and cell borders using classes or ids to make specific selections in the table. For example, a row selector could look like the following:

```
tr.header {background-color: #ccddee;}
```

You would apply this rule using the class attribute in the specific row you want to have the background color:

```
<tr class="header">
```

Figure 10-11 shows a table with the header row styled with a thick blue bottom border and light blue background color.

Best-Selling Albums Worldwide

Artist	Album	Released	Genre	Sales
Michael Jackson	Thriller	1982	Pop	110 mil.
AC/DC	Back in Black	1980	Hard Rock	49 mil.
Pink Floyd	The Dark Side of the Moon	1973	Rock	45 mil.
Whitney Houston	The Bodyguard	1992	Soundtrack	44 mil.
Eagles	Greatest Hits	1976	Rock	42 mil.

Figure 10-11 Table header row with custom bottom border

The style rule for this table shows the selector for the <th> elements with properties that set the border and background color.

```
table {
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
}

th {
   border-bottom: solid thick blue;
   background-color: #ccddee;
}
```

Applying Padding, Margins, and Floats to Tables

See Chapter 6 for more information on the box model properties.

The box properties can be applied to tables to increase spacing within cells, add white space around tables, and float them within blocks of text.

Using Padding

You can enhance the legibility of your table data by adding padding values for the entire table, or alternately adding padding values in individual rows or cells with class selectors. The <table> element does not accept the padding property, so you have to apply it to the <th> and <td> elements. This style rule adds 5 pixels of padding to both types of table elements:

```
th, td {padding: 5px;}
```

The table in Figure 10-12 uses this style rule to apply 5 pixels of padding to every cell.

Figure 10-12 Table with 5 pixels of padding in each cell

```
table {
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}
```

You can also specify individual padding properties for each cell or row. For example, you might specify that <th> cells have extra padding on the bottom:

```
th {padding-bottom: 10px;}
```

Figure 10-13 shows a table with 10 pixels of padding in the header row, and 5 pixels of padding in the data cells.

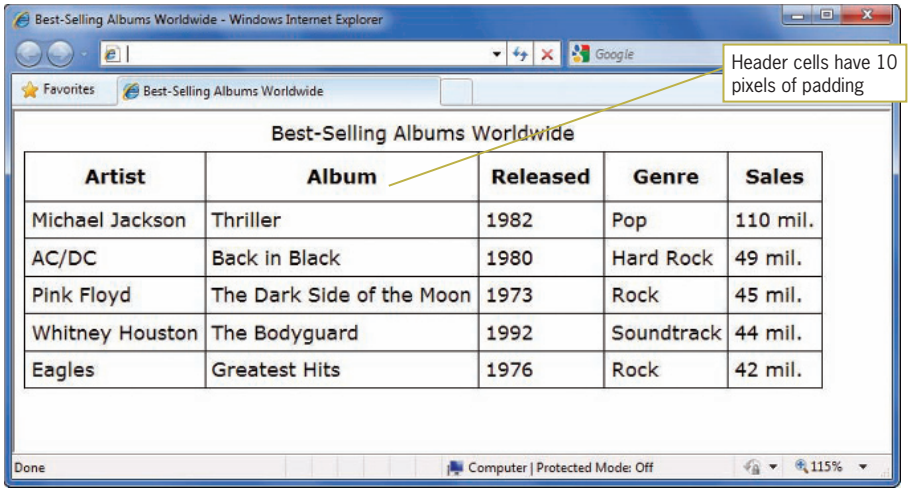


Figure 10-13 Table with 5 pixels of padding in each data cell, 10 pixels in header cell

The style rule has separate selectors for the <th> and <td> elements.

```
table {
   border: solid 1px black;
   border-collapse: collapse;
}

th {
   padding: 10px;
}

td {
   border: solid 1px black;
   padding: 5px;
}
```

Using Margins and Floats

Tables can be floated like any other block-level element. When you float a table next to a block of text, you can provide white space around the table with the margin properties.

Figure 10-14 shows a table floated to the left, with right and bottom margin settings to offset the table from text.

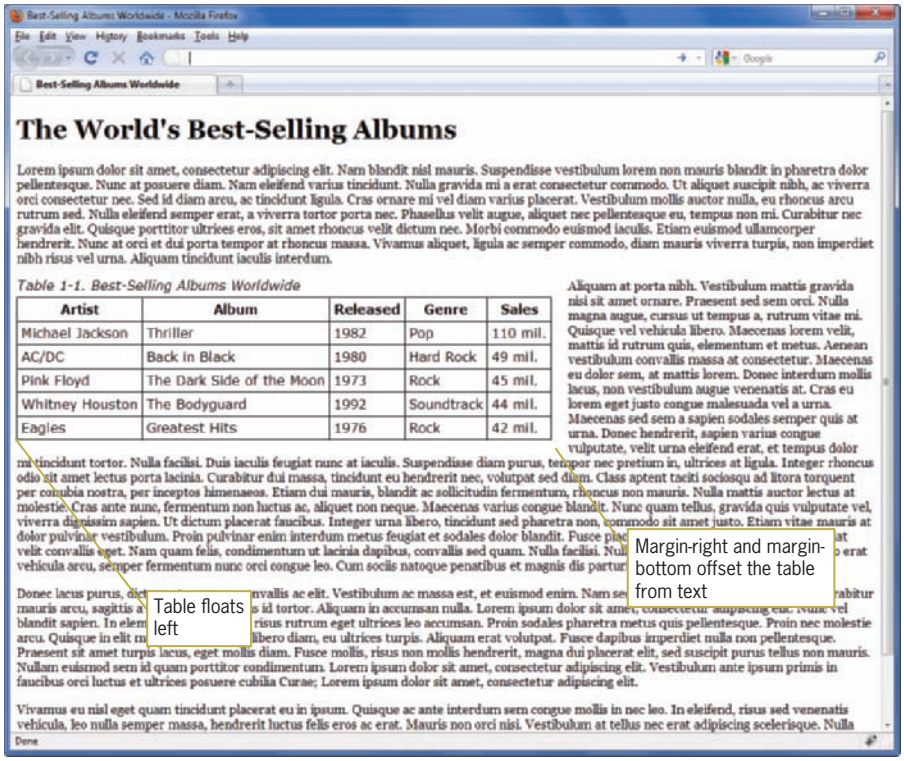


Figure 10-14 Table floated to the left with right and bottom margins

The style rule for this table follows. Notice that this table has a class selector named *best* that specifies only this one table for these style rules. The table has float and margin properties that position it on the page.

```
table.best {
   font-family: verdana;
   border: solid 1px black;
   border-collapse: collapse;
   float: left;
   margin-right: 20px;
   margin-bottom: 10px;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}
```

```
caption {
   padding-bottom: 5px;
   text-align: left;
   font-style: italic;
}
```

Remember that you can select and style individual tables by giving them a class or id name to apply style rules only to the selected table.

Styling Table Background Colors

You can use the background color properties to add legibility to your table data. You can apply background colors to an entire table, single out rows and cells, or use the <colgroup> and <col> properties to highlight individual columns. You can alternate colors for different rows of data, or add hover interaction that highlights data when a user selects rows or cells.

Specifying Background Color

Background colors can make your tables easier to read by providing contrast. Figure 10-15 shows a table with different background colors for the column titles and data. Notice that the text in the column titles is white against the dark blue background.

Artist	Album	Released	Genre	Sales
Michael Jackson	Thriller	1982	Pop	110 mil.
AC/DC	Back in Black	1980	Hard Rock	49 mil.
Pink Floyd	The Dark Side of the Moon	1973	Rock	45 mil.
Whitney Houston	The Bodyguard	1992	Soundtrack	44 mil.
Eagles	Greatest Hits	1976	Rock	42 mil.

Figure 10-15 Styling table colors

The style rules for this table specify a background and text color for the <th> elements and a different background color for the <td> elements.

```
table {
   border: solid 1px black;
   border-collapse: collapse;
   color: #722750
}

td, th {
   border: solid 1px black;
   padding: 5px;
}

caption {
   padding-bottom: 5px;
}

th {
   padding: 10px;
   background-color:#7fa2c1;
   color: white;
}

td {
   background-color: #ccdae6;
}
```

Creating Alternate Color Rows

Table data becomes much easier to read when alternate rows have a distinguishing background color. This effect is easy to create with a class that selects the alternate rows. Write a style rule for the odd (or even) row using a class selector that specifies the class named *odd* applied to the <tr> element as shown. This style rule selects all of the <td> elements within that row to apply the background-color property:

```
tr.odd td {background-color: #eaead5;}
```

Then add the class attribute to every odd row in the table:

```
<tr class="odd">
   <td>AC/DC</td>
   <td>Back in Black</td>
   <td>1980</td>
   <td>Hard Rock</td>
   <td>49</td>
</tr>
```

The result is shown in Figure 10-16.

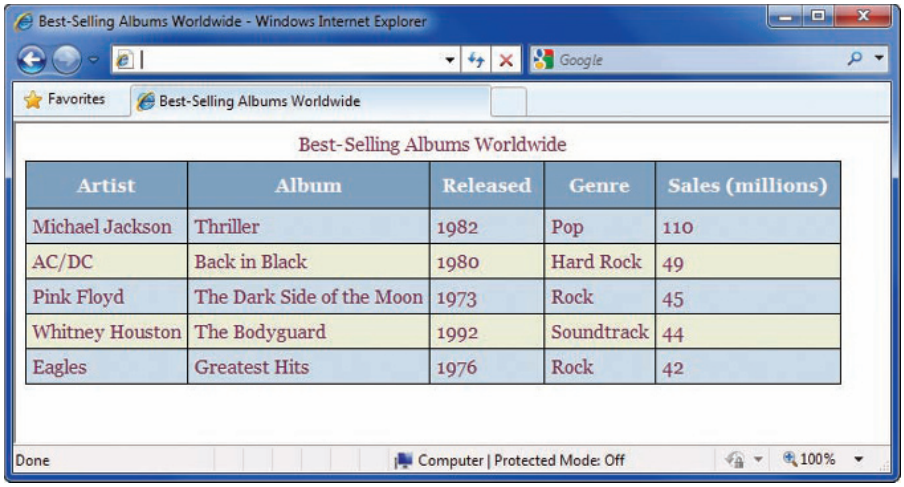


Figure 10-16 Styling alternate row colors

Creating Background Hover Effects

You can add interactivity to your table by adding hover effects. When a user hovers the pointer over a cell or row then the background, font, or formatting can change. Figure 10-17 shows a table with highlighted data activated by the pointer hovering over the cell.

Figure 10-17 Table cell hover

This effect uses the :hover pseudo-class described in Chapter 4. The following style rules state the default cell background color (#ccdae6) and the different background and text color when the user hovers the pointer over a table cell.

```
td {
   background-color: #ccdae6;
}

td:hover {
   color: white;
   background-color: #722750;
}
```

This same effect can be applied to a row of data by applying the hover to <tr> elements as shown. These style rules show the default cell background and the alternate background and text color when the user hovers the pointer over any row of data.

```
td {
   background-color: #ccdae6;
}

tr:hover td {
   color: white;
   background-color: #722750;
}
```

Figure 10-18 shows the result of the row hover.

Figure 10-18 Table row hover

Activity: Applying Table Styles

In the following set of steps, you will style a table using CSS. Save your file and test your work in the browser as you complete each step. Refer to Figure 10-22 as you progress through the steps to see the results. New code that you will add is shown in blue. Save your file and test your work in the browser as you complete each step.

To style the table:

1. Copy the **table project.html** file from the Chapter10 folder provided with your Data Files to the Chapter10 folder in your work folder. (Create the Chapter10 folder, if necessary.)
2. In your browser, open **table project.html.** When you open the file it looks like Figure 10-19.

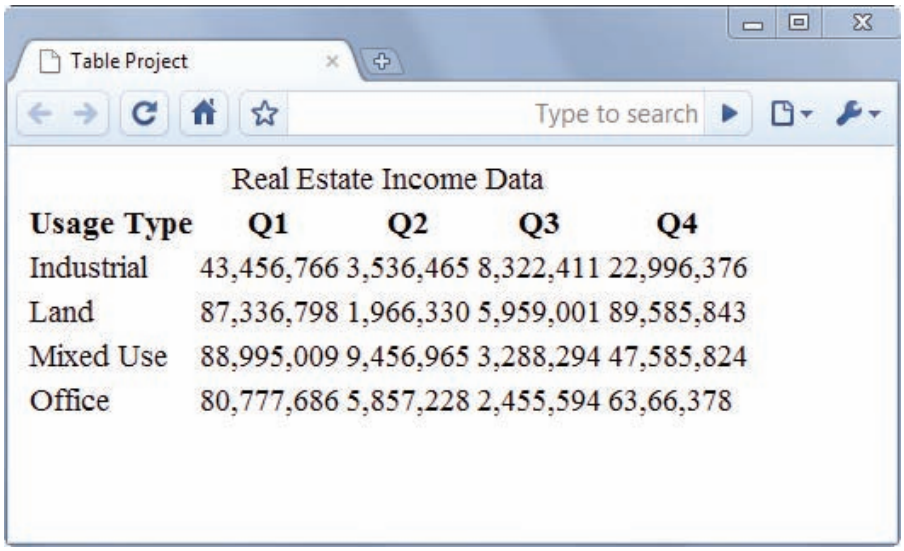


Figure 10-19 Beginning project file

3. Start by setting a font family for the table. Use *table* as a selector, and specify Georgia as the font with an alternate generic font.

```
<style type="text/css">
table {
   font-family: georgia, serif;
}

</style>
```

4. Specify a border for the outside of the table. Collapse the borders of the table as well.

```
<style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

</style>
```

5. Specify a border and padding for both types of cell elements, <th> and <td>.

```
<style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}

</style>
```

6. View the file in the browser. It should look like Figure 10-20.

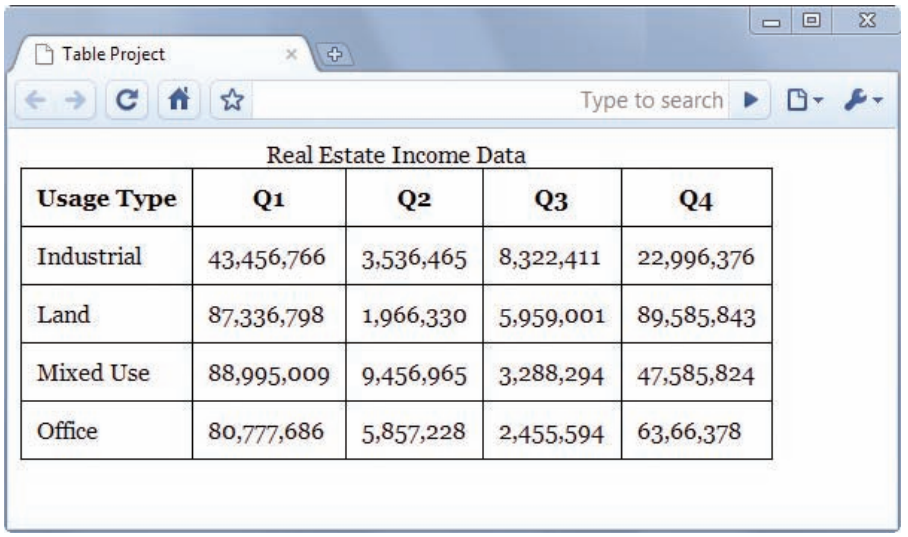


Figure 10-20 Table with basic styling applied

7. Add a background color for the header <th> cells.

```
style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}

th {
   background-color: #d0dafd;
   color: #0070dd;
}

</style>
```

8. Now prepare to add a background color to the far-left column by adding a <colgroup> element to the table code. Find the <caption> element in the table code, and add a <colgroup> element with class and span values as shown. These tags let you select the left column and apply a style to it.

```
<table>
<caption>Real Estate Income Data</caption>
<colgroup class="leftcol" span="1"></colgroup>

...rest of table code...
```

9. Select the colgroup class, and apply a background color style as shown.

```
<style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}

th {
   background-color: #d0dafd;
   color: #0070dd;
}
```

```
colgroup.leftcol {
   background-color: #d0dafd;
}

</style>
```

10. View the file in the browser. It should like Figure 10-21.

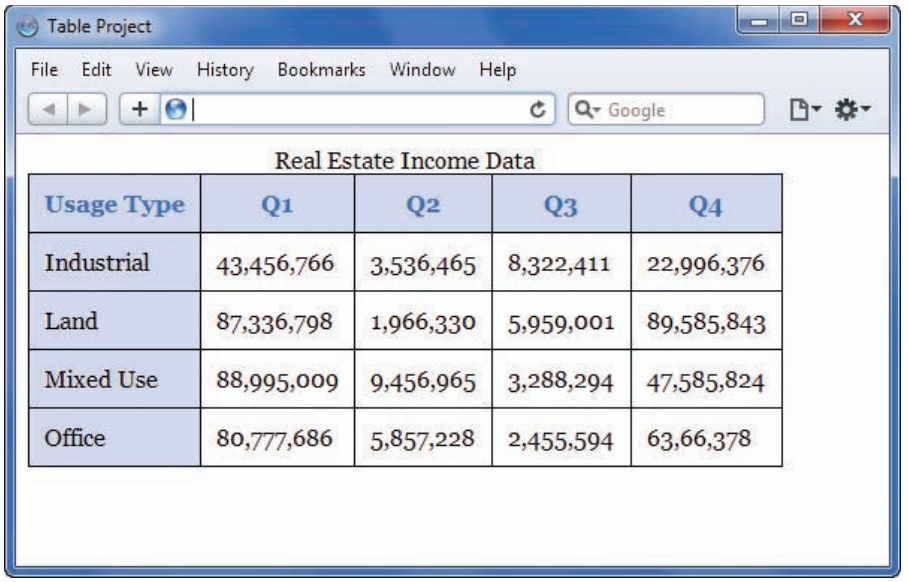


Figure 10-21 Table with background colors applied

11. Add a style for the caption that changes the font-family and adds some bottom padding to move it away from the table border.

```
<style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}

th {
   background-color: #d0dafd;
   color: #0070dd;
}
```

```
colgroup.leftcol {
   background-color: #d0dafd;
}

caption {
   font-family: verdana, sans-serif;
   padding-bottom: 10px;
}

</style>
```

12. Finally, add a hover style that changes the background color when a user hovers over a cell.

```
<style type="text/css">
table {
   font-family: georgia, serif;
   border: solid 1px black;
   border-collapse: collapse;
}

th, td {
   border: solid 1px black;
   padding: 5px;
}

th {
   background-color: #d0dafd;
   color: #0070dd;
}

colgroup.leftcol {
   background-color: #d0dafd;
}

caption {
   font-family: verdana, sans-serif;
   padding-bottom: 10px;
}

td:hover {
   background-color: #ff99ff;
}

</style>
```

13. View the file in the browser. It should like Figure 10-22.

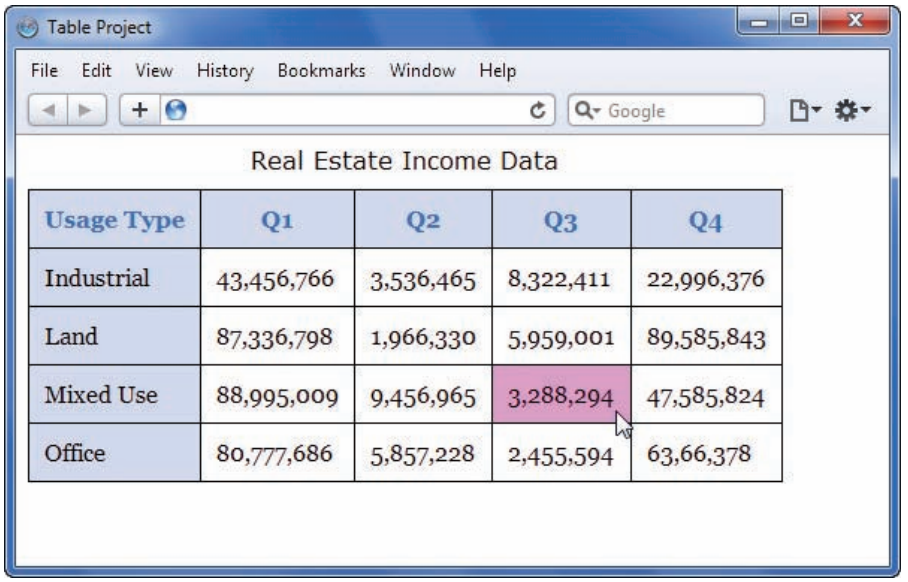


Figure 10-22 Finished styled table

Chapter Summary

- Use tables for presentation of data, not for page layout.
- To build effective data tables, you must be familiar with the HTML table elements including the <table>, <caption>, and <th>, <tr>, and <td> elements.
- Use the grouping elements to apply styles to groups of rows or columns, or to the header, body, and footer of a table.
- Apply borders to both the <table> and cell (<th> and <td>) elements to display a table border on the entire table.
- Use the border-collapse property to make table data more legible.
- Always use CSS to add presentation style to tables.
- Use padding to add space within your cells to make your data more legible.

- You can float tables and add margins with the box model properties.
- Specify background colors or hovers to aid in the legibility of your data.

Key Terms

<caption> element—An HTML element that lets you add a caption to the table.

colspan—An attribute that lets you create cells spanning multiple columns of a table.

rowspan—An attribute that lets you create cells spanning multiple rows of a table.

<table> element—An HTML element that contains table information.

table data cell (<td>)—An HTML element that contains the table data.

table header element (<th>)—An HTML element that contains the table header, which is the heading at the top of a column of data.

table row element (<tr>)—An HTML element that contains the table cells.

Review Questions

1. What are the three basic table elements?
2. What table element presents its content as bold and centered?
3. What table element lets you add a title for a table?
4. What CSS property can you use to change the alignment of the <caption> element?
5. What CSS property do you use to remove spacing between table cells?
6. What are the two table attributes that let you span columns and rows?

7. What value should colspan equal in the following code?

```
<tr><td>R1C1</td><td>R1C2</td><td>R1C3</td></tr>
<tr><td>R2C1</td><td colspan=  >R2C2</td></tr>
```

8. What CSS property do you use to adjust spacing within table cells?

9. What HTML elements let you apply styles to table columns?

10. Which elements do you have to apply borders to if you want borders to be displayed for an entire table?

11. Do border properties inherit from the <table> element to <td> elements?

12. What is one way to select a specific table row or cell?

13. What is a good way to add legibility to rows of data?

Hands-On Projects

1. In this project, you have a chance to apply some of the data table techniques you learned about in this chapter. As you work through the steps, refer to Figure 10-23 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Copy the **project1.html** file from the Chapter10 folder provided with your Data Files to the Chapter10 folder in your work folder. (Create the Chapter10 folder, if necessary.)

   b. Open the file **project1.html.** This is a blank HTML file with just the necessary structural elements such as <head>, <body>, and so on.

   c. Build a simple table with four columns and five rows as shown. Use any data you choose or copy the data supplied here.

   d. Make sure you include the caption and header row.

e. Display the results in a browser, and then compare them to Figure 10-23.

AFI Top 5 Movies

#	Movie	Year	Director
1	Citizen Kane	1941	Orson Welles
2	The Godfather	1972	Francis Ford Coppola
3	Casablanca	1942	Michael Curtiz
4	Raging Bull	1980	Martin Scorsese
5	Singing in the Rain	1952	Stanley Donen

Figure 10-23 Project 1 solution

2. In this project, you continue working in the project1.html file from the previous exercise. As you work through the steps, refer to Figure 10-24 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Add the following style properties to the table in project1.html:

   - All borders collapsed
   - font family: arial
   - table border: solid 4px blue
   - table cells border: solid 1px black
   - table cells padding: 10 pixels

   b. Display the results in a browser, and then compare them to Figure 10-24.

AFI Top 5 Movies

#	Movie	Year	Director
1	Citizen Kane	1941	Orson Welles
2	The Godfather	1972	Francis Ford Coppola
3	Casablanca	1942	Michael Curtiz
4	Raging Bull	1980	Martin Scorsese
5	Singing in the Rain	1952	Stanley Donen

Figure 10-24 Project 2 solution

3. In this project, you continue working in the project1.html file from the previous exercise. As you work through the steps, refer to Figure 10-25 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Style the table caption with the following properties:

      - Alignment: Left-aligned
      - Font size: .85em
      - Font style: italic
      - Padding: 10 pixels on the bottom

   b. Display the results in a browser, and then compare them to Figure 10-25.

AFI Top 5 Movies

#	Movie	Year	Director
1	Citizen Kane	1941	Orson Welles
2	The Godfather	1972	Francis Ford Coppola
3	Casablanca	1942	Michael Curtiz
4	Raging Bull	1980	Martin Scorsese
5	Singing in the Rain	1952	Stanley Donen

Figure 10-25 Project 3 solution

4. In this project, you continue working in the project1.html file from the previous exercise. As you work through the steps, refer to Figure 10-26 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Add a background color (#929292) and font-color (#fff) to the table header cells.

   b. Display the results in a browser, and then compare them to Figure 10-26.

AFI Top 5 Movies

#	Movie	Year	Director
1	Citizen Kane	1941	Orson Welles
2	The Godfather	1972	Francis Ford Coppola
3	Casablanca	1942	Michael Curtiz
4	Raging Bull	1980	Martin Scorsese
5	Singing in the Rain	1952	Stanley Donen

Figure 10-26 Project 4 solution

5. In this project, you continue working in the project1.html file from the previous exercise. As you work through the steps, refer to Figure 10-27 to see the results you will achieve. Save your file and test your work in the browser as you complete each step.

   a. Add a background hover color (#66ccff) when the user points to a row of data.

   b. Display the results in a browser, and then compare them to Figure 10-27.

AFI Top 5 Movies

#	Movie	Year	Director
1	Citizen Kane	1941	Orson Welles
2	The Godfather	1972	Francis Ford Coppola
3	Casablanca	1942	Michael Curtiz
4	Raging Bull	1980	Martin Scorsese
5	Singing in the Rain	1952	Stanley Donen

Figure 10-27 Project 5 solution

Individual Case Project

Examine the content you are presenting in your project Web site and find data that would be enhanced by the use of a table. Write a brief, single-page analysis or memo that you can present to your instructor that explains why you should or should not use tables to present your data. If your analysis warrants a table, then design and implement the table in the appropriate Web page(s).

Team Case Project

As a team, examine the content you are presenting in your project Web site and find data that would be enhanced by the use of a table. Write a brief, single-page analysis or memo that you can present to your instructor that explains why you should or should not use tables to present your data. If your analysis warrants a table, then design and implement the table in the appropriate Web page(s).

CHAPTER 11

Web Forms

When you complete this chapter, you will be able to:

- Understand how forms work
- Use the <form> element
- Create input objects
- Style forms with Cascading Style Sheets (CSS)
- Build a form

This chapter covers the HTML form elements. Forms let you build interactive Web pages that collect information from a user and process it on the Web server. You can use forms to gather information and create databases or to send customized responses to your users. Forms collect—but do not process—data. The data processing must be performed on the Web server that hosts the form. Forms are the basis for online commerce; without them users would not be able to enter customer address, credit card, and ordering information on the Web.

Understanding How Forms Work

Figure 11-1 shows a typical form. You can use a variety of input elements for the form based on the type of information you want to gather from your user. Forms usually contain basic HTML formatting tags such as <p> and <br/>. Forms can also be styled with CSS, which helps control their visual layout. Well-designed forms include active white space, aligned form elements, and clear labels. Use the design principles you have learned throughout this book to create forms that are legible and easy to use.

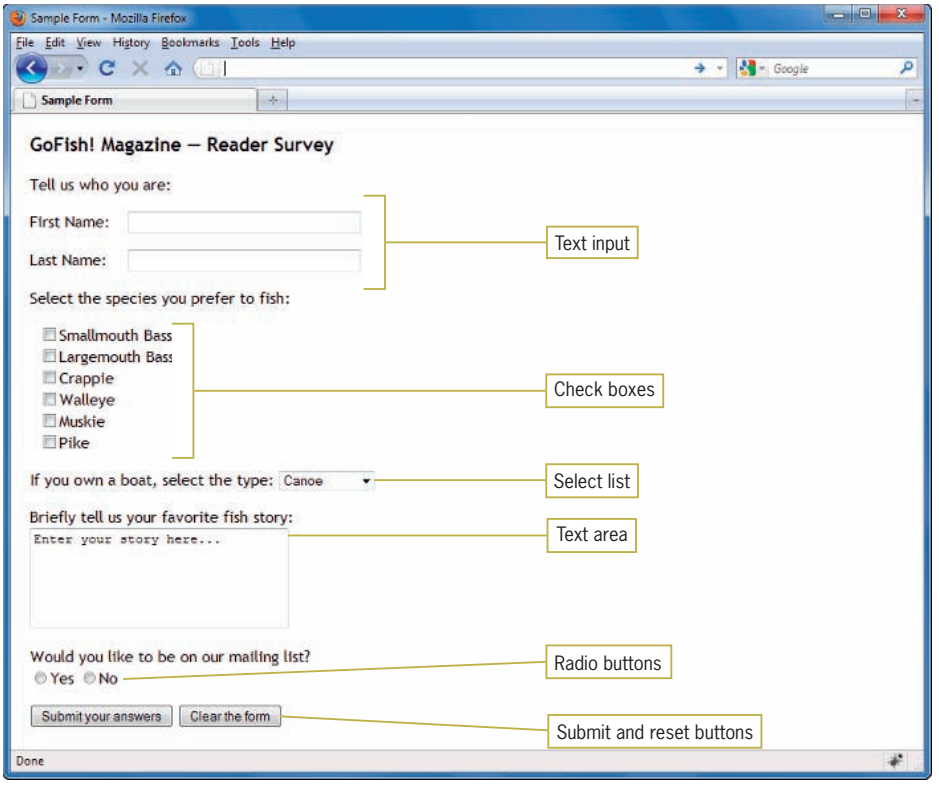


Figure 11-1 Sample HTML form

The HTML form itself is the interface for the user to enter data, but all of the actual data processing is performed on the server using applications that reside on the Web server or in the Common Gateway Interface (CGI). The traditional method of processing forms input has been to use the **Common Gateway Interface**, which is the communications bridge between the Internet and the server. Using programs called scripts, CGI can collect data sent by a user via the Hypertext Transfer Protocol (HTTP) and transfer it to a variety of data-processing programs, including spreadsheets, databases, or other software running on the server. The data-processing software can work with the data and send a response back to CGI, and then on to the user.

Although CGI is still in use, there are alternatives that are faster and more efficient. Most of the popular Web server software, such as Apache on Linux Web servers and Internet Information Service (IIS) on Windows Web servers, have their own plug-in modules that run forms-processing software.

The programs that manage the processing of the collected data can be written in a variety of programming languages, although scripting languages are the most common. The scripting language most commonly used with HTML forms is **JavaScript**, which was originally created for the Netscape browser in 1995. JavaScript is a client-side scripting language, which means that it runs on the user's computer, rather than on the server. JavaScript can enhance your Web site with beneficial programming functions, including form validation that checks a user's form entries for errors before the result is sent to a form **script**, which is a program that transfers form data to a server. A complete JavaScript tutorial is out of the scope of this book, though many good references are available on the Web

Good sources for JavaScript information on the Web include the following: *www.yourhtmlsource.com/javascript* *www.w3schools.com/js/default.asp*

A more recent enhancement to forms processing is **AJAX**, which is short for Asynchronous JavaScript and XML. AJAX is a group of technologies that is used on the client to retrieve data from the user and submit the server request in the background. The result is that the data can be processed without the user having to wait for the page to reload, allowing enhanced speed and interactivity. Despite the name AJAX, scripts can be created with languages other than JavaScript and does not need XML to manage the data. You can find out more about scripting and AJAX at *www.w3.org/standards/webdesign/script*.

Using the <form> Element

The <form> element is the container for creating a form, as the <table> element is the container for the elements that create a table. A form has a number of attributes that describe how the form data is handled, as described in Table 11-1.

Attribute	Description
action	The URL of the application that processes the form data; this URL points to a script file or an e-mail address
enctype	The content type used to submit the form to the server (when the value of the method is "post"); most forms do not need this attribute
method	Specifies the HTTP method used to submit the form data; the default value is "get" • get—The form data is appended to the URL specified in the action attribute • post—The form data is sent to the server as a separate message
accept	A comma-separated list of content types that a server processing this form can handle correctly; most forms do not need this attribute
accept-charset	A list of allowed character sets for input data that is accepted by the server processing this form; most forms do not need this attribute

Table 11-1 Form Attributes

The <form> element by itself does not create a form. It must contain **form controls** (such as <input> elements) and structural elements such as <p> or <ul> to control the look of the form. CSS style classes let you select different form elements and control their look with style rules.

A variety of form controls are available for collection information, as described in the following sections. The following code shows a typical <form> element with some of the attributes listed in Table 11-1. This code specifies that the form data the user enters is being sent to a program named register.asp that resides on a Web server.

```
<form method="post"
action="https://signup.website.com/register.asp">
```

Using get or post

The method you will specify in your form is often defined by the programmers that create the script that process the form data. The difference between *get* and *post* is the way the data is sent to the server.

```
method="get"
```

This method sends the form information by including it in the URL. The data is not secure and should not be used for passwords or other confidential information.

```
method="post"
```

This method sends the form information securely to the server within the message body, so the data is not visible. This is the more common method for sending form data.

Using the mailto Action

With the mailto action, you can collect data from a form and send it to any e-mail address. Although this method does not allow any data processing, it is fine for the occasional brief form or for simple Web sites. The data is sent as a long string of text, which you can make easier to read by including the enctype="text/plain" attribute. A form element with a mailto action looks like the following:

```
<form action="mailto:joel@joelsklar.com"
method="post" enctype="text/plain">
```

The data will be sent as an e-mail message to the specified e-mail address. For example, the data from an address form would look like this:

name=Joe User
street=3 Maple Lane
city=Smalltown
state=MA
zip=00000

The mailto action depends on the user having e-mail client software configured on his or her system. If a mail client is configured, the user's e-mail software will open when the user clicks the submit button. If no mail client is configured, the user will not be able to submit the form data.

Creating Input Objects

The <input> element defines many of the form input object types. Table 11-2 lists the available object types. You use the *type* attribute to specify the object type.

Type Attribute Value	Description
text	Creates a text entry field that lets the user enter a single word or a line of text; this is the default object type
password	Creates the same type of text entry field created by the value "text," but the user entry is masked by asterisks
checkbox	Provides on/off toggles that the user selects; check boxes are best used with multiple-answer questions, and multiple check boxes can contain the same name, letting you group them together so that users can select multiple values for the same property.
radio	Lets a user choose one value from a range of values; when radio buttons are grouped together with the same name, only one choice can be selected
submit	Sends the form data to the server using the transmission method specified in the <form> element; every form needs a submit button
reset	Clears the form of any user-entered data and returns it to its original state
hidden	Adds a control that is not displayed in the browser; the hidden type is useful for sending additional information with the form data that may be needed for processing
image	Adds a graphic button to the form, rather than the default button
button	Creates a button that has no default behavior; the button's function is usually defined by a script; when the user pushes the button, the script function is triggered
file	Lets the user select a file that is submitted with the form

Table 11-2 <input> Element Types

HTML5 offers new form input types that are currently not supported by most browsers. You will find descriptions of these new form input types in Appendix A.

Labeling Form Elements

The <label> element lets you create a caption for an input element. Figure 11-2 shows the label element next to both text and check box elements.

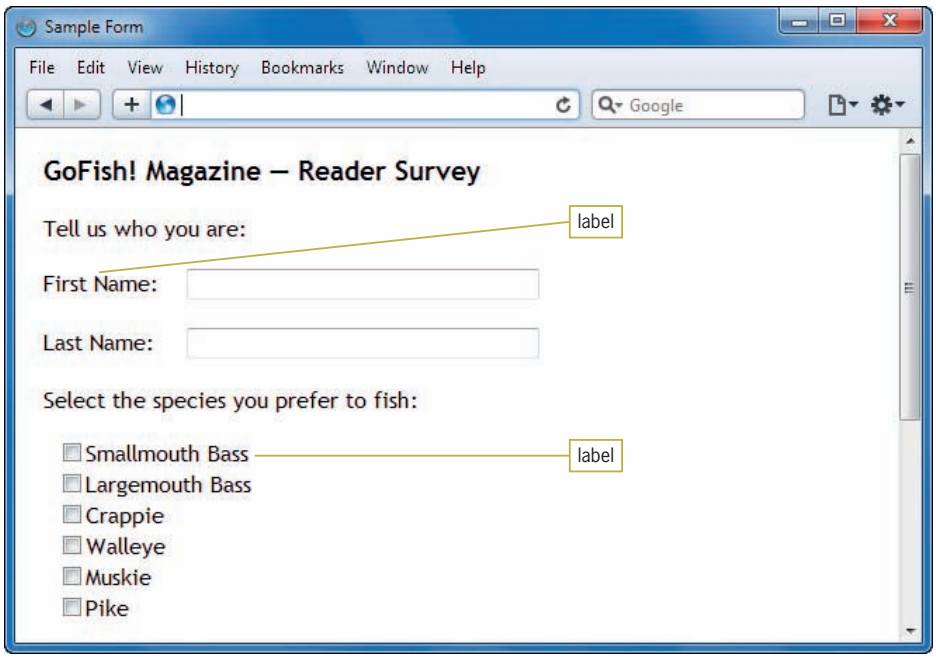


Figure 11-2 The <label> element provides captions in a form

The <label> element in the following code sample contains the caption text *First Name:*.

```
<p>
   <label class="username" >First Name:</label>
   <input type="text" name="firstname"
    size="35" maxlength="35" />
</p>
```

The <label> element lets you extend the clickable area of a form element to make them more accessible. For example, rather than having to click only in the check box, users can also click the text caption to select the check box as shown in Figure 11-3.

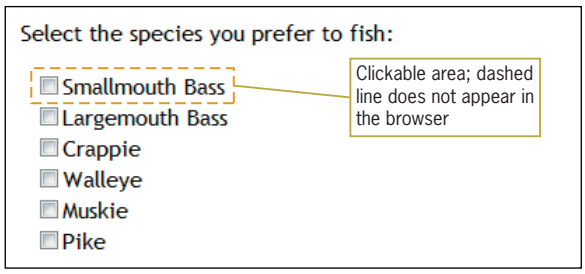


Figure 11-3 The <label> element increases the clickable area

To make the text clickable, you associate the <label> element with the <input> element by using the *for* and *id* attributes as shown in the following code.

```
<p>
   <label class="username" for="First Name">
   First Name:</label>
   <input type="text" name="firstname" id="First Name"
   size="35" maxlength="35" />
</p>
```

The *for* attribute in the <label> element must match the *id* attribute in the <input> element to associate the text with the input element.

Creating Text Boxes

The text entry box is the most commonly used <form> input type. The default text box is 20 characters long, although you can change this value using the *size* attribute. The user can enter an unlimited number of characters in the text box even though they exceed the visible length. You can constrain the user's entry of text with the *maxlength* attribute and supply a default value for the text with the *value* attribute. The following code shows a simple form with two text boxes.

```
<form action="http://server.com/cgi_bin/script.cgi"
method="post">

<p>Tell us who you are:</p>

<p>
   <label class="username" for="First Name">
   First Name: </label>
   <input type="text" name="firstname" id="First Name"
   size="35" maxlength="35" /></p>
<p>
   <label class="username" for="Last Name">
   Last Name: </label>
   <input type="text" name="lastname" id="Last Name"
   size="35" maxlength="35" /></p>

</form>
```

This code creates the two text box inputs shown in Figure 11-4.

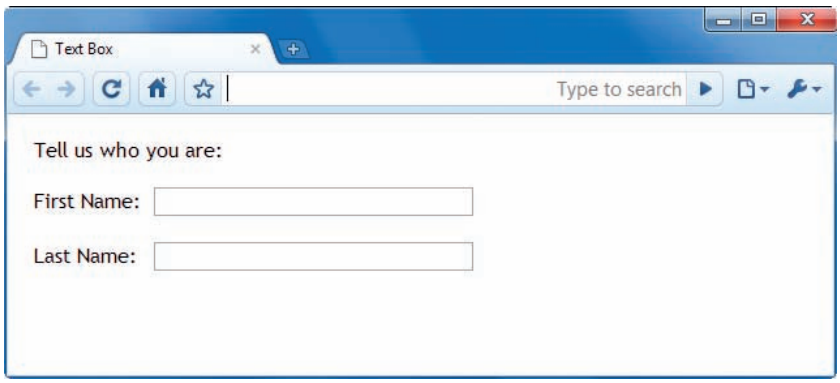


Figure 11-4 Text box input type

Creating Check Boxes

Check boxes are on/off toggles that the user can select. You can use the name attribute to group check boxes, allowing the user to select multiple values for the same property. Figure 11-5 shows an example of a group of check boxes.

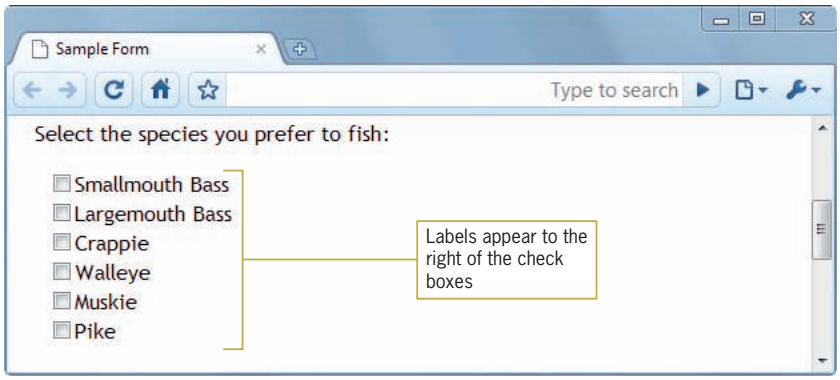


Figure 11-5 Check box input type

In the following code, the various fish species check boxes are grouped together with the name attribute set to *species*. Notice that the check boxes reside within a standard unordered list with the CSS list-style-type property set to *none*. Also, the <label> element comes after the <input> element to display the captions to the right of the check boxes.

```
<ul>
<li>
<input type="checkbox" name="species" value="smbass"
id="Smallmouth Bass" />
<label for="Smallmouth Bass">Smallmouth Bass</label>
</li>

<li>
<input type="checkbox" name="species" value="lgbass"
id="Largemouth Bass" />
<label for="Largemouth Bass">Largemouth Bass</label>
</li>

<li>
<input type="checkbox" name="species" value="crappie"
id="Crappie" />
<label for="Crappie">Crappie</label>
</li>

<li>
<input type="checkbox" name="species" value="walleye"
id="Walleye" />
<label for="Walleye">Walleye</label>
</li>

<li>
<input type="checkbox" name="species" value="muskie"
id="Muskie" />
<label for="Muskie">Muskie</label>
</li>

<li>
<input type="checkbox" name="species" value="pike"
id="Pike" />
<label for="Pike" />Pike</label>
</li>
</ul>
```

To check a check box by default, you can use the checked attribute. The following code fragment shows the syntax for this attribute. Here, the Pike check box is checked by default.

```
<input type="checkbox" name="species" value="pike"
checked="checked" />Pike
```

Creating Radio Buttons

Radio buttons are like check boxes, but only one selection is allowed. When radio buttons are grouped with the *name* attribute, only one value can be selected to be "on," while all other values must be "off." To preselect one of the radio buttons, you use the checked attribute. Figure 11-6 shows the radio buttons input type.

Figure 11-6 Radio buttons input type

In the following code, the *Yes* and *No* radio buttons are grouped together with the name attribute set to *list*. The user can choose only one of the two values. The <label> elements provide the captions next to the button.

```
<p>Would you like to be on our mailing list?</p>
<p><input type="radio" name="list" value="yes"
id="Yes" />
<label for="Yes">Yes</label>
<input type="radio" name="list" value="no"
id="No" />
<label for="No">No</label>
</p>
```

Use check boxes when you want to create a question to which multiple answers are allowed. Use radio buttons when you want users to choose only one answer.

Creating Submit and Reset Buttons

The submit and reset button input types let the user either send the form data to be processed or clear the form and start over. These are predefined functions that are activated by the button type. Set the input type to either *submit* or *reset*. The default button text values are *Submit Query* and *Reset*. You can use the *value* attribute to customize the button text. Figure 11-7 shows the buttons.

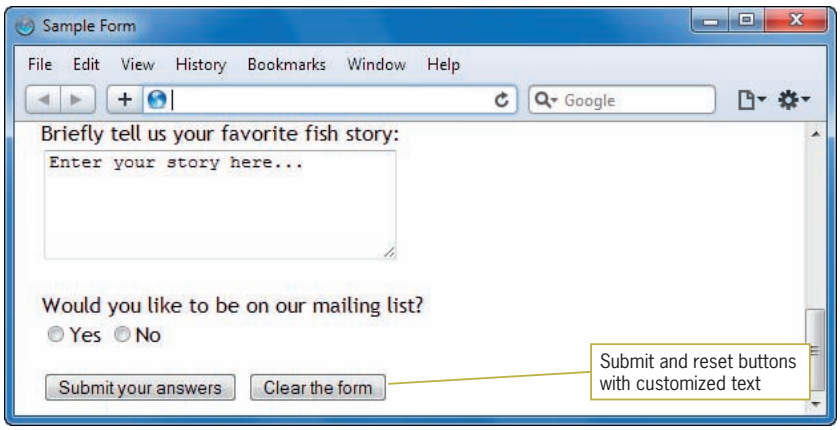


Figure 11-7 Submit and reset input buttons

The following code shows the addition of submit and reset buttons with customized button text shown in the figure.

```
<p>
<input type="submit" value="Submit your answers" />
<input type="reset" value="Clear the form" />
</p>
```

Creating an Image for the Submit Button

You can choose an image file and use it instead of the default button image for the submit button. The image type works only for the submit function. Make sure that the image you choose is an acceptable Web file format (GIF, PNG, or JPG). The src attribute contains the location of the image file. Remember to include an alt attribute as you would with any other image. Figure 11-8 shows the use of an image (submit.jpg) for the submit button.

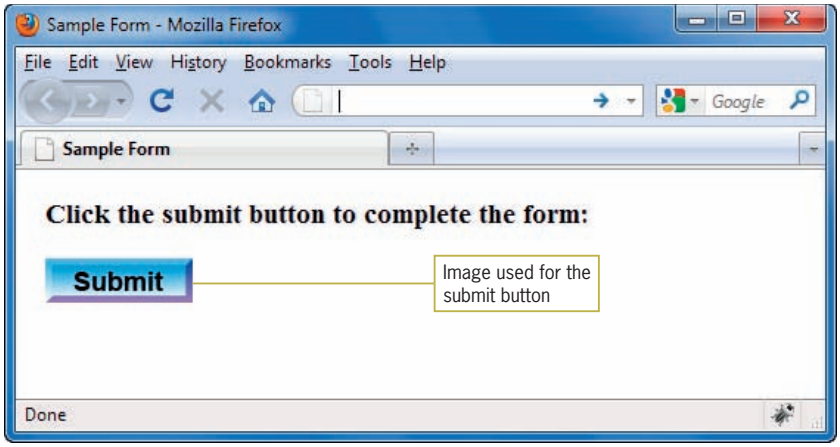


Figure 11-8 Using an image for the submit button

The following code shows the use of an image input type specifying the image and alt attribute.

```
<h3>Click the submit button to complete the form:</h3>
<p><input type="image" src="submit.jpg"
alt="submit button" /></p>
```

Letting the User Submit a File

The file input type object lets users select a file on their own computers and send it to the server. This type lets you create a text input area for the user to enter a filename. The length of the text input is specified with the size attribute. The file type automatically includes a browse button that lets users browse for a file on their computer. This input type looks different in different browsers, as shown in Figure 11-9, which includes both Internet Explorer and Safari results.

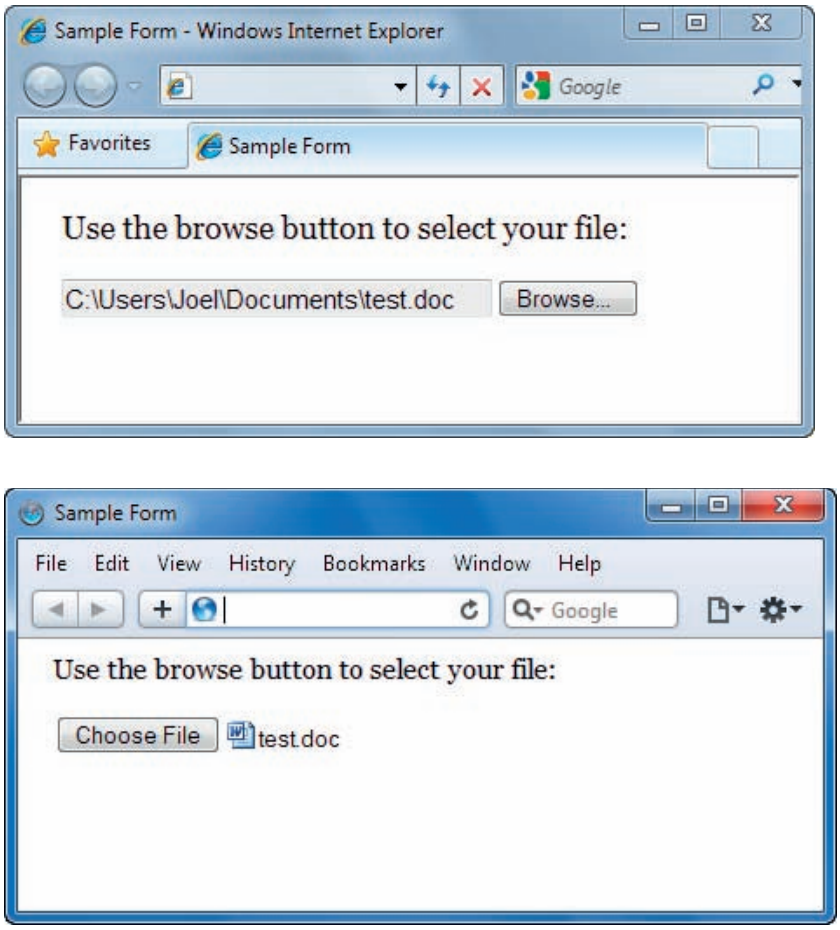


Figure 11-9 File input type in Internet Explorer (top) and Safari (bottom)

The following code shows the file input type. In this case, the text box accepts up to 30 characters.

```
<p>Use the browse button to select your file:</p>
<p><input type="file" size="30" /></p>
```

Creating a Password Entry Field

The password input type object works like a text input box, with the additional feature that the entered text is hidden by asterisks rather than shown on the screen. This is a very low level of password protection, as the password is only protected from unauthorized users looking at the screen. The password itself is sent to the server as plain text, and anyone with network access could read the password information. If you use passwords, check with your system administrator to see whether you can send passwords over a secure Internet connection. Figure 11-10 shows password input type.

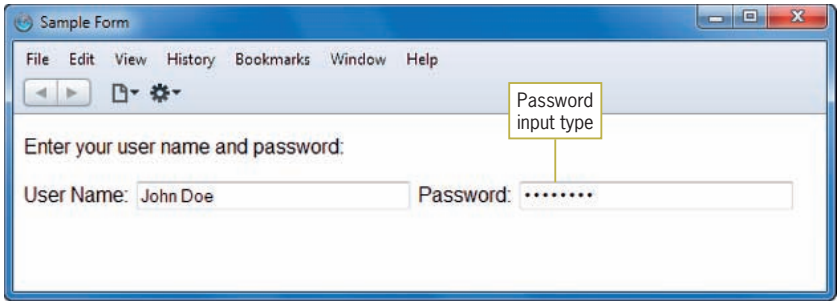


Figure 11-10 Password input type

The following code shows the use of the password input type. Both the user name and password text boxes accept up to 30 characters.

```
<p>Enter your user name and password:</p>
<p>
User Name: <input type="text" size="30" />
Password: <input type="password" size="30" />
</p>
```

Using the <select> Element

The <select> element lets you create a list box or scrollable list of selectable options. The <select> element is a container for the <option> element. Each <option> element contains a list value.

The following code shows the standard type of list box; the user can choose one value from the list. Figure 11-11 shows the result of the code.

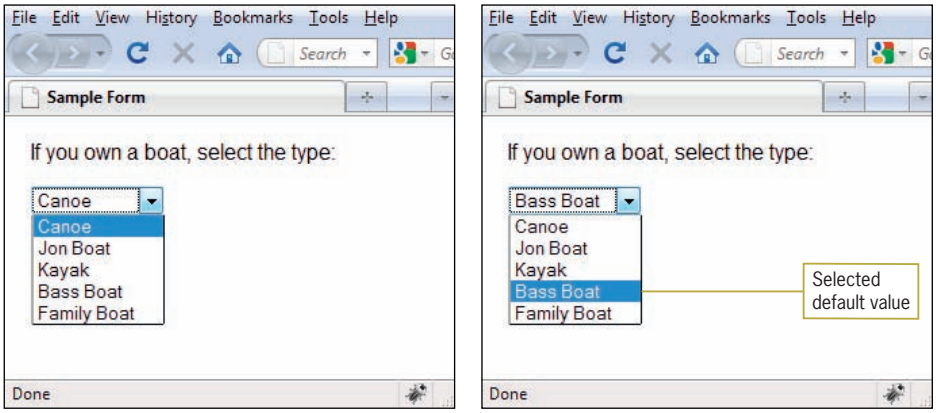


Figure 11-11 Select list element

Notice that the first option in the list is the value that appears in the list box text area.

```
<p>If you own a boat, select the type:</p>
<p>
<select name="boats">
   <option value="canoe">Canoe</option>
   <option value="jonboat">Jon Boat</option>
   <option value="kayak">Kayak</option>
   <option value="bassboat">Bass Boat</option>
   <option value="familyboat">Family Boat</option>
</select>
</p>
```

You can select the default value in a list by adding the *selected* attribute to an <option> element. In the following list, "Bass Boat" is the default value, as shown in the browser window on the right in Figure 11-11.

```
<p>If you own a boat, select the type:</p>
<p>
<select name="boats">
   <option value="canoe">Canoe</option>
   <option value="jonboat">Jon Boat</option>
   <option value="kayak">Kayak</option>
   <option value="bassboat" selected="selected">
   Bass Boat</option>
   <option value="familyboat">Family Boat</option>
</select>
</p>
```

You can also let the user pick multiple values from the list by adding the multiple attribute to the <select> element. The user can hold the Ctrl key and select multiple items in the list, or hold the Shift key to select contiguous options. Figure 11-12 and the following code show the use of the multiple attribute.

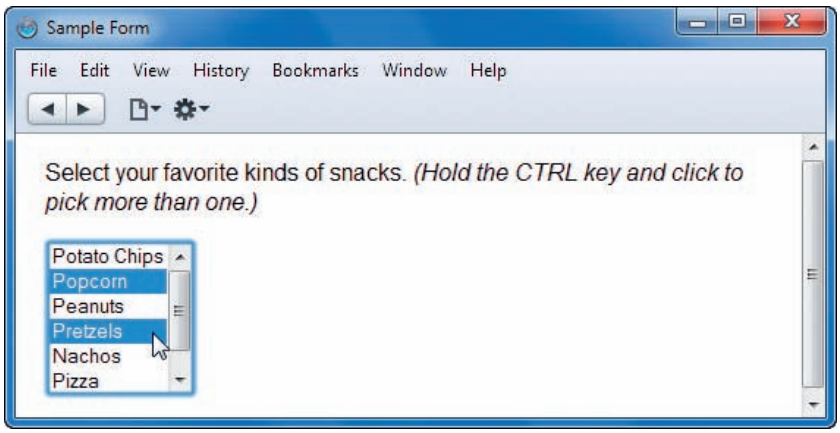


Figure 11-12 Scrollable select list with multiple choices

The size attribute specifies how many list options are visible at a time. The following list shows six options at once.

```
<p>Select your favorite kinds of snacks. <em>(Hold
the CTRL key and click to pick more than one.)</em></p>
<p>
<select name="snacks" multiple size="6">
   <option>Potato Chips</option>
   <option>Popcorn</option>
   <option>Peanuts</option>
   <option>Pretzels</option>
   <option>Nachos</option>
   <option>Pizza</option>
   <option>Fries</option>
</select>
</p>
```

Grouping List Options

You can group and label sets of list options with the <optgroup> element and label attribute. The result is a heading for a series of options within a list. Figure 11-13 shows the result of using the <optgroup> element.

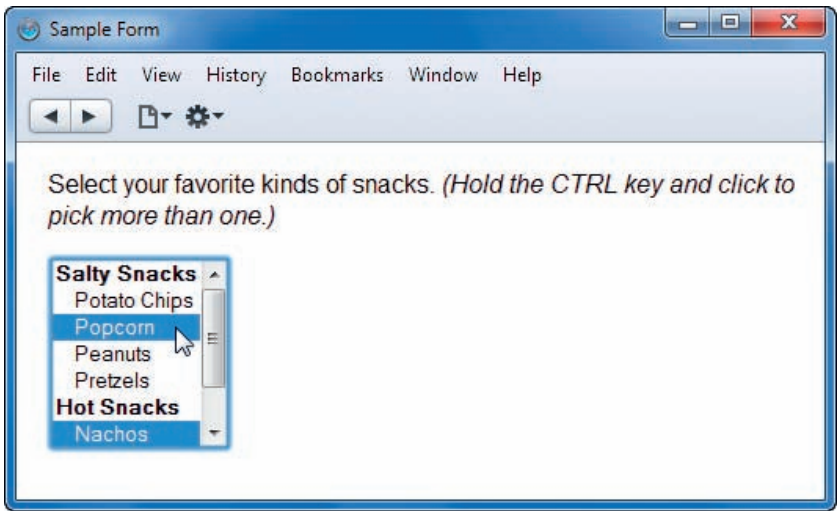


Figure 11-13 Grouping list options

The browser determines the format of the labels, but they are usually displayed in bold text. In the following code, the snack list is divided into two groups: salty snacks and hot snacks.

```
<p>
<select name="snacks" multiple size="7">
<optgroup label="Salty Snacks">
   <option>Potato Chips</option>
   <option>Popcorn</option>
   <option>Peanuts</option>
   <option>Pretzels</option>
</optgroup>
<optgroup label="Hot Snacks">
   <option>Nachos</option>
   <option>Pizza</option>
   <option>Fries</option>
</optgroup>
</select>
</p>
```

Using the <textarea> Element

The <textarea> element lets you create a text area for user input larger than the <input> text type object described previously. Figure 11-14 shows a sample of a <textarea> element.

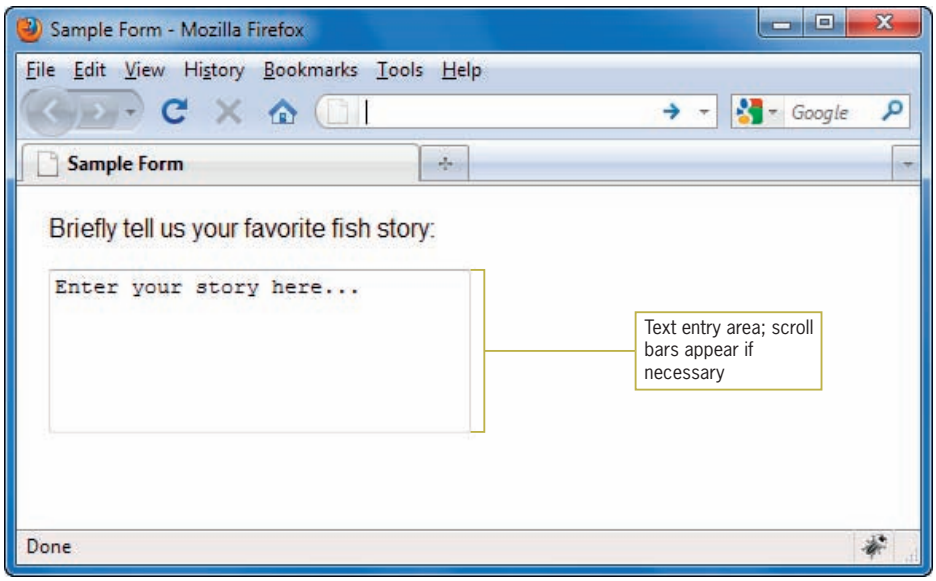


Figure 11-14 Using the <textarea> element

You can specify the width and height of the text area with the *cols* and *rows* attributes. If the text entered is longer than the height allotted, the browser adds scroll bars automatically. Any text you enter in the <textarea> element appears as the default text in the user's browser. The following code shows a text area set to 30 columns wide by 5 rows high.

```
<p>Briefly tell us your favorite fish story:</p>
<p>
<textarea name="fishstory" rows="5" cols="30">
Enter your story here...
</textarea>
</p>
```

Creating Input Groupings

You can use the <fieldset> and <legend> elements to create groupings of different types of <input> elements. The <fieldset> element contains the <input> elements, and the <legend> element contains a label for the grouping. These two elements help make your forms more readable and increase their accessibility to screen readers and other accessibility devices. Figure 11-15 shows the use of the <fieldset> and <legend> elements. The code for the page follows.

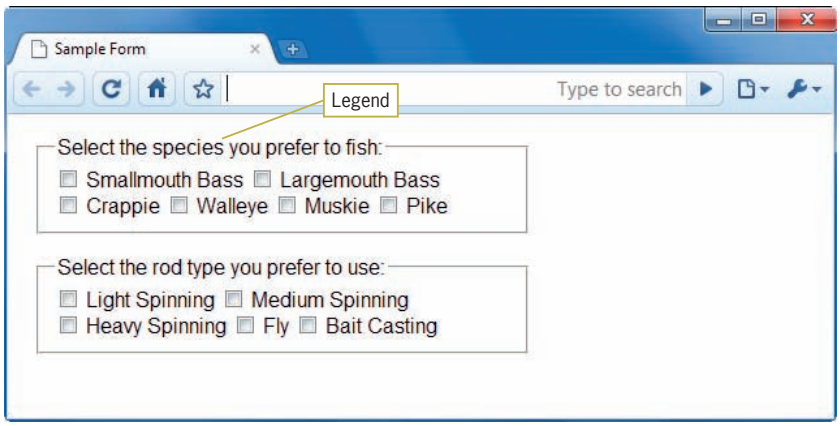


Figure 11-15 Grouping and labeling <input> elements

```
<fieldset>
<legend>Select the species you prefer to fish:
</legend>
<p>
<input type="checkbox" name="species"
value="smbass" />
Smallmouth Bass
<input type="checkbox" name="species"
value="lgbass" />
Largemouth Bass <br/>
<input type="checkbox" name="species"
value="crappie" />
Crappie
<input type="checkbox" name="species"
value="walleye" />
Walleye
<input type="checkbox" name="species"
value="muskie" />
Muskie
<input type="checkbox" name="species"
value="pike" />
Pike
</p>
</fieldset>

<fieldset>
<legend>Select the rod type you prefer to use:
</legend>
<p>
<input type="checkbox" name="species"
value="ltspin" />
Light Spinning
<input type="checkbox" name="species"
value="mdspin" />
Medium Spinning <br/>
<input type="checkbox" name="species"
```

```
value="hvspin" />
Heavy Spinning
<input type="checkbox" name="species"
value="fly" />
Fly
<input type="checkbox" name="species"
value="btcas" />
Bait Casting
</p>
</fieldset>
```

Styling Forms with CSS

Most forms can benefit from CSS styling to increase their legibility and appeal. As you have seen in the form samples in this chapter, forms need at least basic formatting elements, such as <br/> and <p>, to place form elements on separate lines and add white space. Even with these basic formatting elements, the look of your form may not be acceptable. Figure 11-16 shows a typical form. Notice how the left justification of the form elements gives a ragged look to the form.

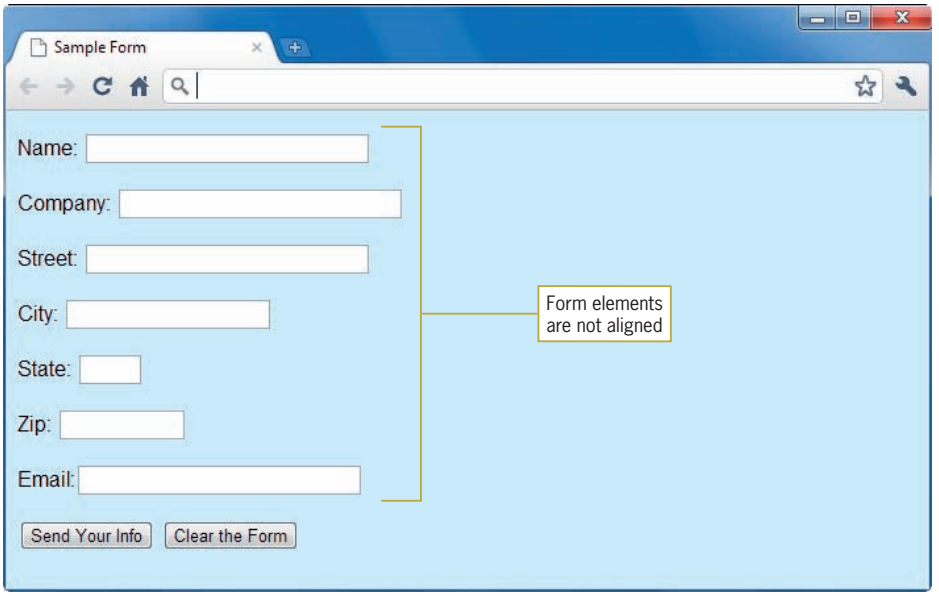


Figure 11-16 Typical form layout

In contrast to Figure 11-16, the form in Figure 11-17 has been styled with CSS, producing a more visually appealing form that is easier for the user to follow when entering data.

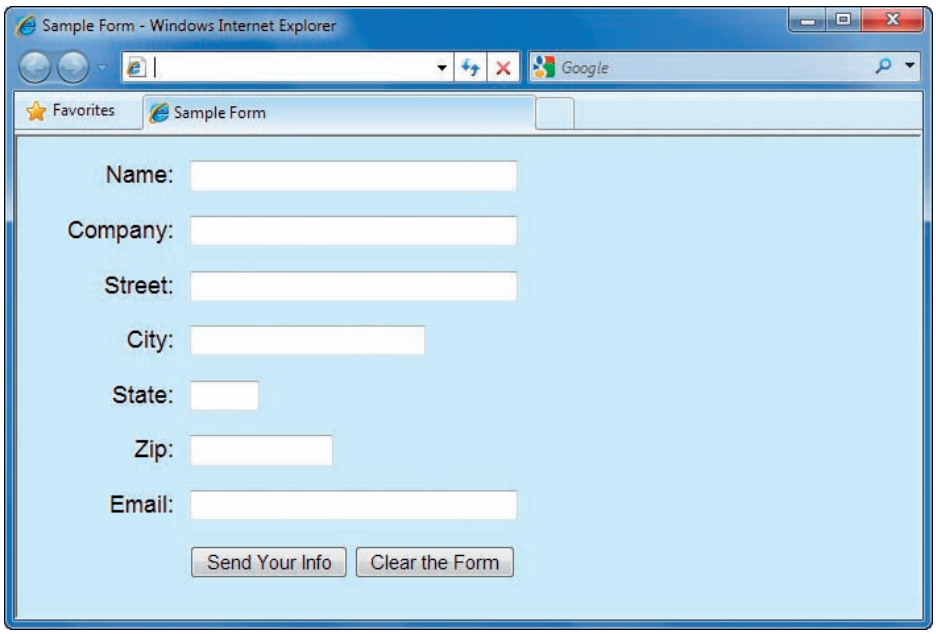


Figure 11-17 Form layout enhanced with CSS

Adding CSS to form elements is no different than adding styles to standard content elements. You can use many of the CSS properties to specify type styles, background colors, box properties, and colors to enhance the look of your forms. The grouping and labeling elements <fieldset>, <legend>, and <label> are useful when applying styles to forms. You can also use classes or ids to apply styles to specific form elements.

Aligning Form Elements

As shown in Figure 11-17, aligned form elements create a more organized and easy-to-use form. To create this alignment, you can set a width for the labels so they align in a column to the left of the form input elements. First, examine the code that creates the form in Figure 11-17.

```
<form method="post" action="http://someserver/
cgi_bin/script.cgi">
<p>
<label for="name">Name:</label>
<input type=text size=30 maxlength=256 name="Name"
id="name" />
</p>
```

```
<p>
<label for="company">Company:</label>
<input type=text size=30 maxlength=256
name="company" id="company" />
</p>

<p>
<label for="street">Street:</label>
<input type=text size=30 maxlength=256
name="street" id="street" />
</p>

<p><label for="city">City:</label>
<input type=text size=20 maxlength=256
name="city" id="city" />
</p>

<p>
<label for="state">State:</label>
<input type=text size=2 maxlength=256
name="state" id="state" />
</p>

<p>
<label for="zip">Zip:</label>
<input type=text size=10 maxlength=256
name="zip" id="zip" />
</p>

<p>
<label for="email">Email:</label>
<input type=text size=30 maxlength=256
name="email" id="email" />
</p>

<p>
<input class="submit" type="submit"
value="Send Your Info" />
<input type="reset" value="Clear the Form" />
</p>
</form>
```

Notice that the label and each corresponding form element are contained within a <p> element. You can float the label to the left of the form element within each paragraph. Use the label element as the selector for the style rule. Float the label to the left and set a width that fits the label content as shown:

```
label {
   float: left;
   width: 6em;
}
```

Figure 11-18 shows the result of this style rule. In this figure, a border shows the width and placement of the <label> element.

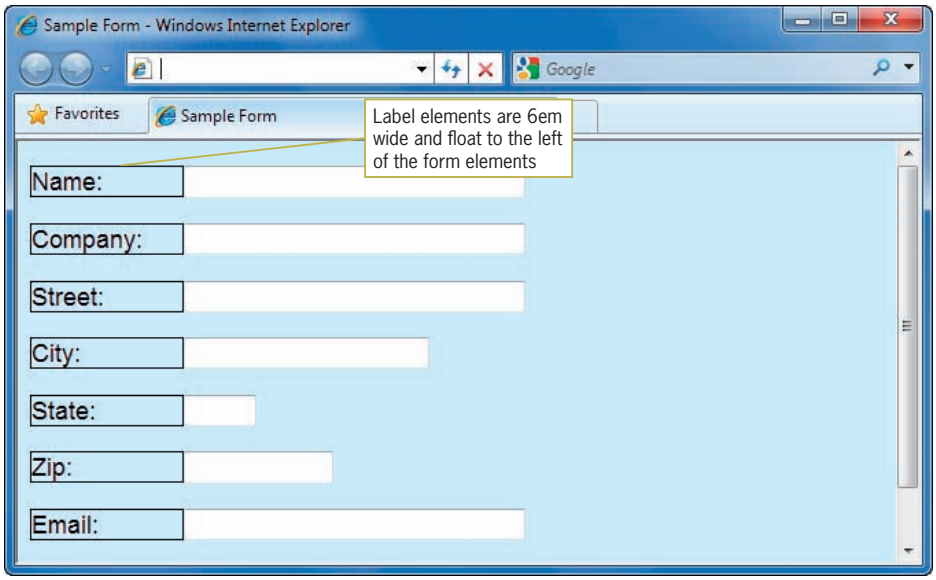


Figure 11-18 Label elements with width and float properties applied

To complete the labels, add style properties to right-align the text and add a right margin to separate the label from the form input element.

```
label {
   float: left;
   width: 6em;
   text-align: right;
   margin-right: 10px;
}
```

Create a style for the paragraph that contains the submit and reset buttons. This rule uses *submit* as the class name and offsets the buttons from the left margin by 8ems.

```
.submit {margin-left: 8em;}
```

Then apply this rule to the paragraph that contains the submit and reset buttons.

```
<p class="submit" >
<input type="SUBMIT" value="Send Your Info" />
<input type="RESET" value="Clear the Form" />
</p>
```

These style rules result in the finished form shown in Figure 11-19.

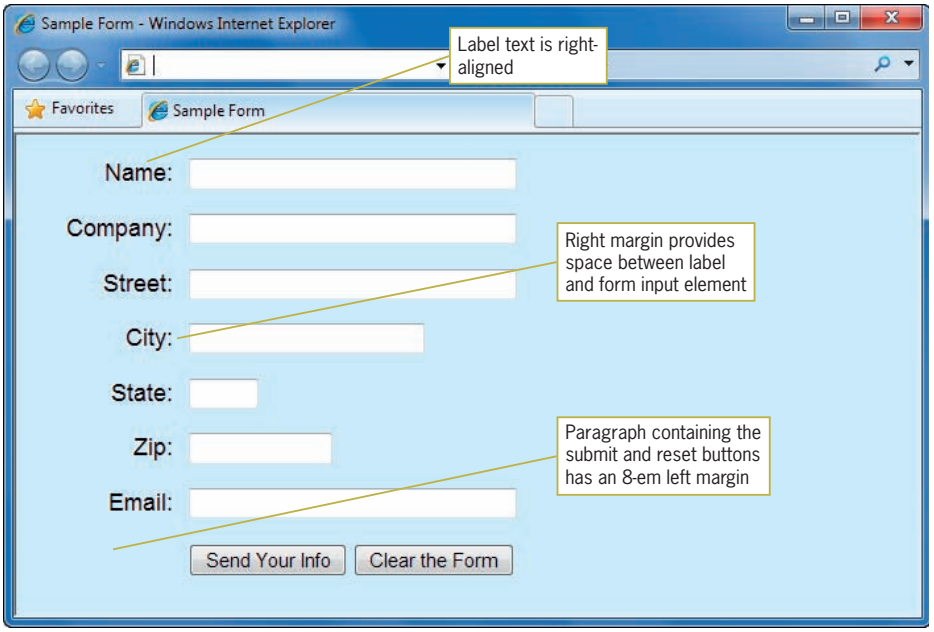


Figure 11-19 Form layout enhanced with CSS

Using an em measurement value for the width makes sure that the label boxes increase in size if the user changes their screen font size.

Styling Fieldset and Legend Elements

The <fieldset> and <legend> elements are great for applying styles to make your forms more attractive. Figure 11-20 shows the form with <fieldset> and <legend> elements with their default display behavior. Notice that the fieldset border extends to the width of the browser window. Also, this form already has the labels and form input elements aligned as you saw in the previous section.

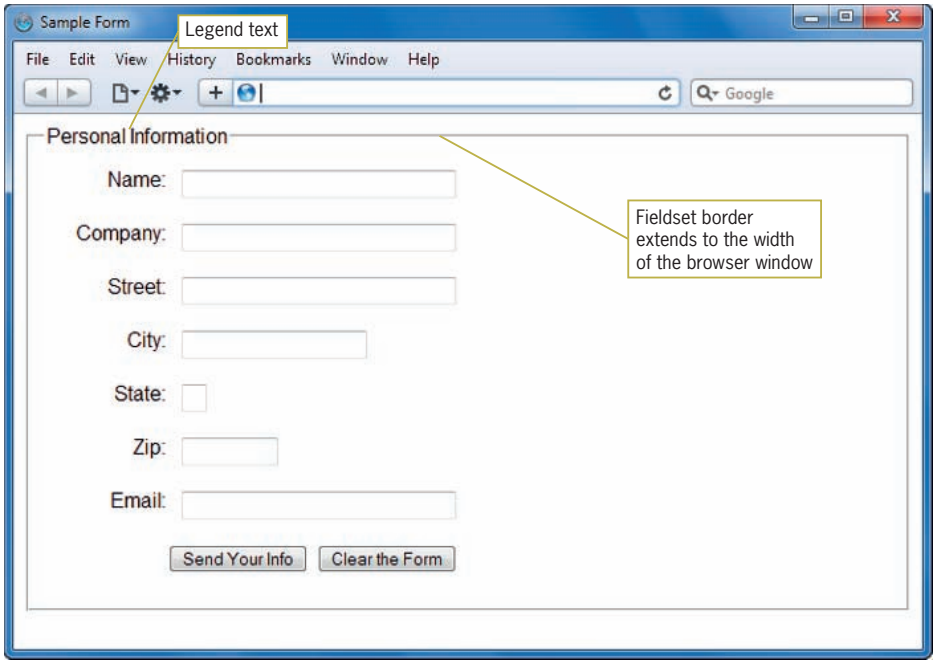


Figure 11-20 Fieldset and legend elements default display

The fieldset can be set to a measured width to keep it from extending to the width of the browser window. The following rule selects the <fieldset> element and applies a width of 24em.

```
fieldset {
   width: 24em;
}
```

Adding a background color (#ddeeff) and a dark blue border (#053972) will help the form stand out.

```
fieldset {
   width: 24em;
   background-color: #ddeeff;
   border: solid medium #053972;
}
```

The legend text can be styled to make it stand out. In this example, the legend had a 1px border that matches the color of the fieldset border. A white background and 2 pixels of padding help offset the text from the fieldset border.

```
legend {
   border: solid 1px #053972;
   background-color: #fff;
   padding: 2px;
}
```

Figure 11-21 shows the results of the fieldset and legend style rules.

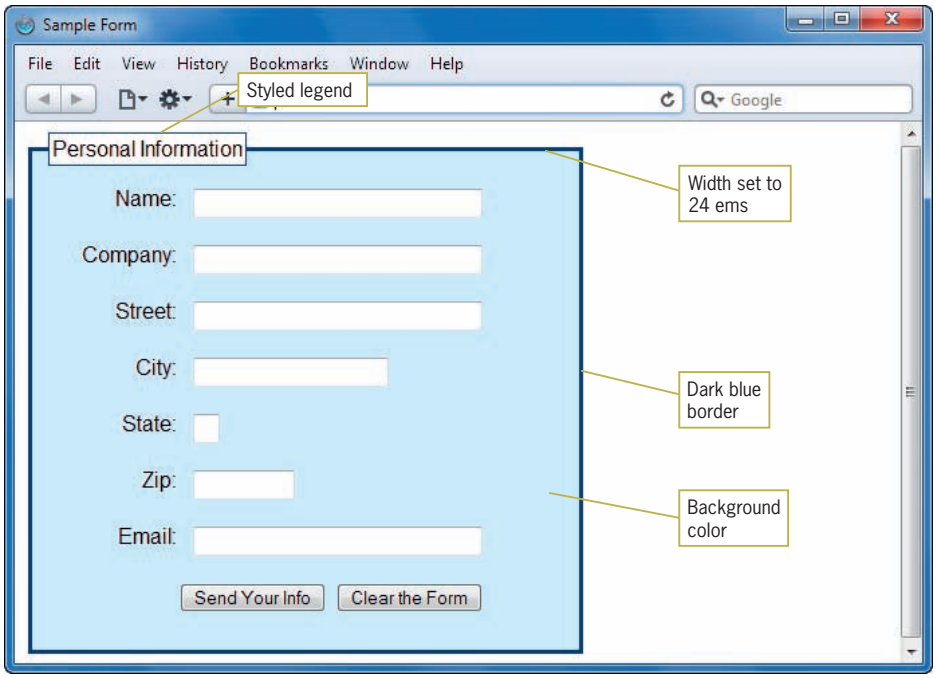


Figure 11-21 Using fieldset and legend style rules

Adding a Background Image

You can add a background image to a fieldset with the background-image property as shown in Figure 11-22.

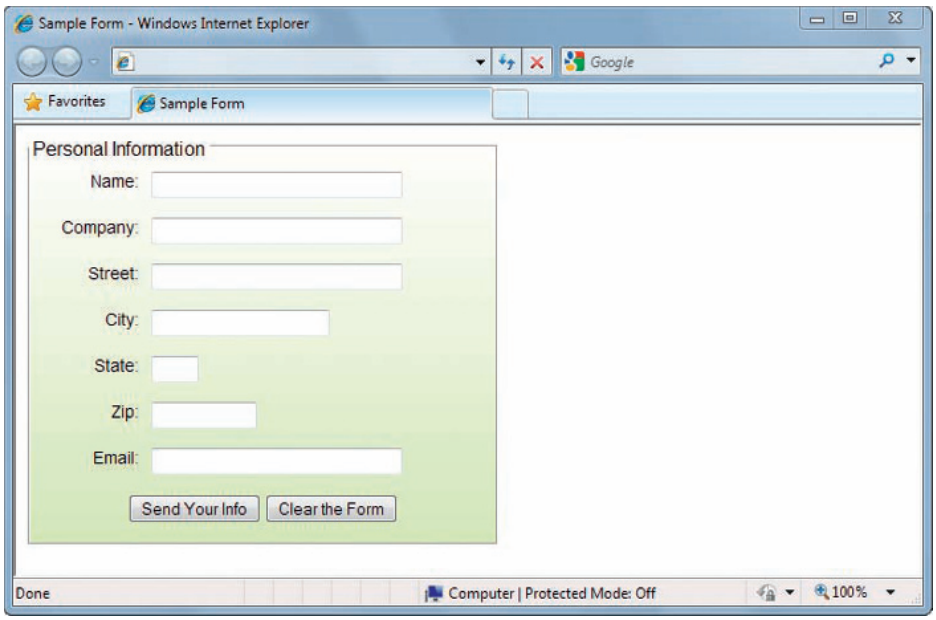


Figure 11-22 Gradient background in a form

In this example, the gradient image (formgradient.jpg) is specified with the background-image property. The background-position is set to bottom, and the background-repeat is set to repeat the graphic along the horizontal axis of the form.

```
fieldset {
   width: 24em;
   background-image: url(formgradient.jpg);
   background-position: bottom;
   background-repeat: repeat-x;
}
```

Activity: Building a Form

In the following set of steps, you will build a form for an online job search service. Users of the service will enter address and personal information into the form.

To begin building the form:

1. Copy the file **form.html** from the Chapter11 folder provided with your Data Files to the Chapter11 folder in your work folder. (Create the Chapter11 work folder, if necessary.)
2. Open **form.html** in your HTML editor, save it as **wonderform.html**, and then examine the code. Note

that a style section contains a body style that sets the font family and left margin for the page.

```
<!DOCTYPE HTML>

<html>
<head>
<title>Wonder Software Online Job Search Form</title>
<meta content="text/html; charset=utf-8"
   http-equiv="Content-Type" />
<style type="text/css">
body {font-family: arial, sans-serif; margin-left: 20px;}
</style>
</head>
<body>
<h1>Wonder Software<br/>Online Job Search</h1>
<form action=" " method="post">
</form>
</body>
</html>
```

3. Begin building the form by adding three text <input> elements, one each for the user's name, e-mail address, and telephone number. Set the *size* and *name* attribute values as shown in the following code. Make sure each input element is contained within a <p> element.

```
<p>Name: <input size="30" name="name" id="name"></p>
<p>Email: ><input size="30" name="email" id="email"></p>
<p>Phone: <input size="30" name="phone" id="phone"></p>
```

4. Add label elements for each text box's label. Make sure to associate the label with the form element by adding the *for* attribute that matches the form input's *id* value.

```
<p><label for="name">Name:</label><input size="30"
   name="name" id="name"></p>
<p><label for="email">Email:</label><input size="30"
   name="email" id="email"></p>
<p><label for="phone">Phone:</label><input size="30"
   name="phone" id="phone"></p>
```

5. Group this set of fields with a <fieldset> and accompanying <legend> element, as shown in the following code.

```
<fieldset>
<legend>Contact Information</legend>
<p><label for="name">Name:</label><input size="30"
   name="name" id="name"></p>
<p><label for="email">Email:</label><input size="30"
   name="email" id="email"></p>
<p><label for="phone">Phone:</label><input size="30"
   name="phone" id="phone"></p>
</fieldset>
```

6. Save **wonderform.html** and leave it open for the next set of steps. Then view the file in the browser; it should now look like Figure 11-23.

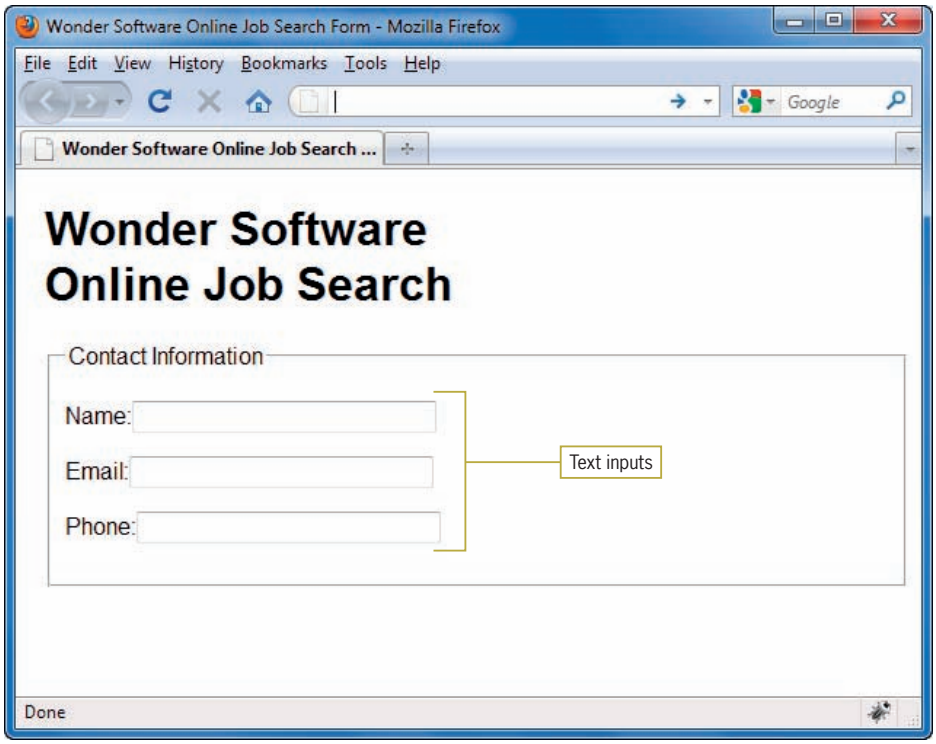


Figure 11-23 Form with three text <input> elements

Adding Check Boxes

Continue to build the form by adding check box <input> elements to collect information from the user. You will also add a list box of job title types, where users can make only one selection from the list.

To continue building the form:

1. Continue working in the file **wonderform.html.**
2. Add text to identify the check box inputs.

```
Select Your Area(s) of Interest:
```

3. Add the check box inputs as shown. The *name* attribute groups the check boxes together under the category *jobtitle*. Each check box is identified with a unique value.

The check boxes are contained in two <p> elements, the first with two job titles and the second with three.

```
<p>
<input type="checkbox" name="jobtitle" value="ae" />
Account Executive

<input type="checkbox" name="jobtitle" value="bd" />
Business Development
</p>

<p>
<input type="checkbox" name="jobtitle" value="is" />
Inside Sales

<input type="checkbox" name="jobtitle" value="sm" />
Sales Manager

<input type="checkbox" name="jobtitle" value="vp" />
VP Sales
</p>
```

4. Add label elements for each check box.

```
<p><input type="checkbox" name="jobtitle" value="ae" />
<label>Account Executive</label>

<input type="checkbox" name="jobtitle" value="bd" />
<label>Business Development</label></p>

<p><input type="checkbox" name="jobtitle" value="is" />
<label>Inside Sales</label>

<input type="checkbox" name="jobtitle" value="sm" />
<label>Sales Manager</label>

<input type="checkbox" name="jobtitle" value="vp" >
<label>VP Sales</label></p>
```

5. Associate the labels with each checkbox input element by adding *for* and *id* attributes to the <label> and <input> elements.

```
<p><input type="checkbox" name="jobtitle" value="ae"
id="ae" />
<label for="ae">Account Executive</label>
```

```
<input type="checkbox" name="jobtitle" value="bd"
id="bd" />
<label for="bd">Business Development</label></p>

<p><input type="checkbox" name="jobtitle" value="is"
id="is" />
<label for="is">Inside Sales</label>

<input type="checkbox" name="jobtitle" value="sm"
id="sm" />
<label for="sm">Sales Manager</label>

<input type="checkbox" name="jobtitle" value="vp"
id="vp" />
<label for="vp">VP Sales</label></p>
```

6. Group this set of fields with a <fieldset> and accompanying <legend> element, as shown in the following code.

```
<fieldset>
<legend>Select Your Area(s) of Interest:</legend>

<p><input type="checkbox" name="jobtitle" value="ae"
   id="ae" /><label for="ae">Account Executive</label>

<input type="checkbox" name="jobtitle" value="bd" id="bd" />
   <label for="bd">Business Development</label></p>

<p><input type="checkbox" name="jobtitle" value="is"
   id="is" /><label for="is">Inside Sales</label>

<input type="checkbox" name="jobtitle" value="sm"
   id="sm" /><label for="sm">Sales Manager</label>

<input type="checkbox" name="jobtitle" value="vp"
   id="vp"> <label for="vp">VP Sales</label></p>

</fieldset>
```

7. Save **wonderform.html** and leave it open for the next set of steps. Then view the file in the browser; it should now look like Figure 11-24.

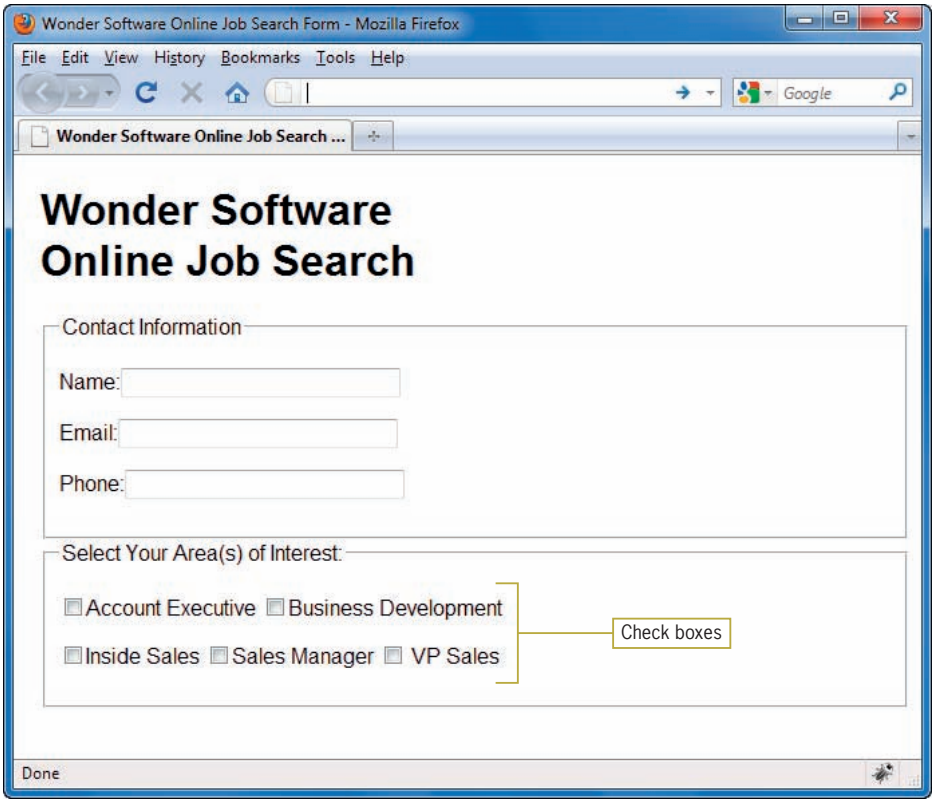


Figure 11-24 Form with check box <input> elements

Adding a List Box and Radio Buttons

Continue to build the form by adding two more <input> elements to collect information from the user. You will add a list box of job position options and a question with a yes or no answer.

To continue building the form:

1. Continue working in the file **wonderform.html.**
2. Add a <select> element with four blank <option> tags, as shown in the following code. Place this code after the closing <fieldset> tag from the previous procedure.

```
<p>
Select the type of position you desire:
<select name="position">
<option> </option>
<option> </option>
<option> </option>
```

```
<option> </option>
</select>
</p>
```

3. Fill in a value for each option.

```
<p>
Select the type of position you desire:
<select name="position">
<option>Part-time contract</option>
<option>Full-time contract</option>
<option>Part-time permanent</option>
<option>Full-time permanent</option>
</select>
</p>
```

4. Below the select list, add the <p> element with the following question:

```
<p>
Are you willing to relocate?
</p>
```

5. Add two <input> elements with the type set to "radio" to create radio buttons.

```
<p>
Are you willing to relocate?
Yes <input type="radio" />
No <input type="radio" />
</p>
```

6. Add a value attribute for each element. Set the value for the Yes button to *yes*. Set the No button to *no*. Also, add a name attribute that groups the radio buttons together with a value of *relocate*.

```
<p>
Are you willing to relocate?
Yes <input type="radio" value="yes" name="relocate" />
No <input type="radio" value="no" name="relocate" />
</p>
```

7. Add labels with *for* and *id* attributes.

```
<p>
Are you willing to relocate?
<label for="yes">Yes</label><input type="radio"
   value="yes" name="relocate" id="yes" />
<label for="no">No</label><input type="radio"
   value="no" name="relocate" id="no"/>
</p>
```

8. Save **wonderform.html** and leave it open for the next set of steps. Then view the file in the browser; it should now look like Figure 11-25.

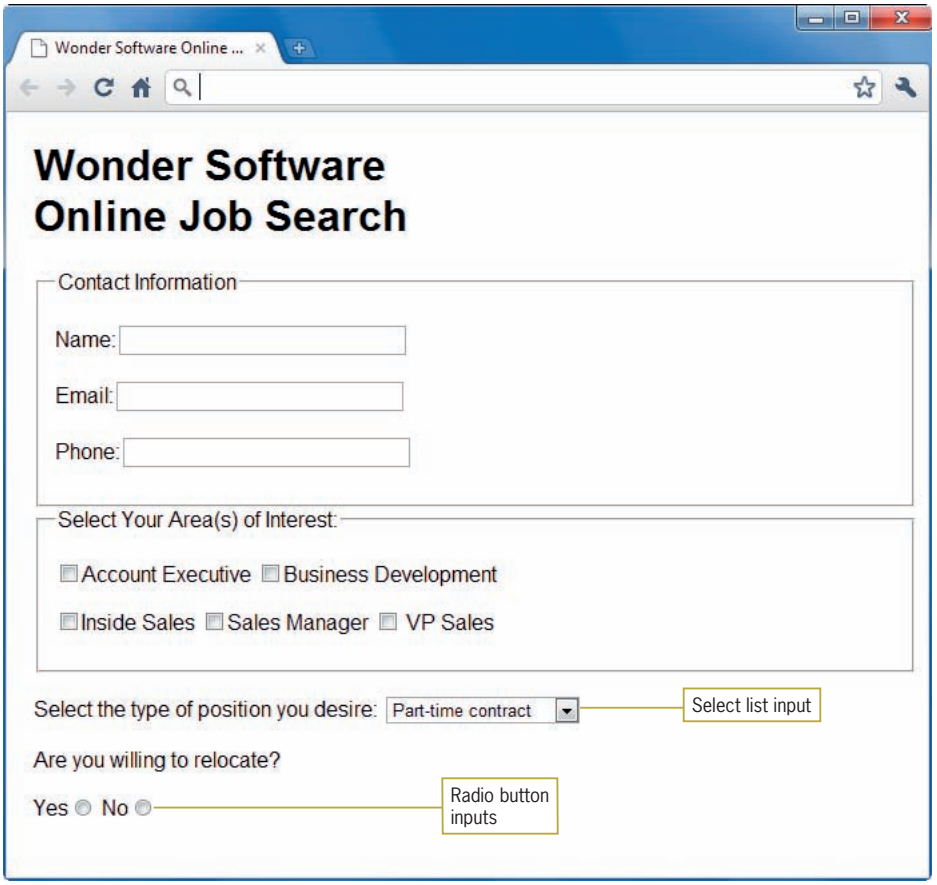


Figure 11-25 Adding a select list and radio buttons

Adding Submit and Reset Buttons

You finish the form by adding the submit and reset buttons.

To continue building the form:

1. Continue working in the file **wonderform.html**.
2. Add submit and reset button element types, and set values for each button as shown.

```
<p>
<input type="submit" value="Submit" />
<input type="reset" value="Clear" />
</p>
```

3. Save **wonderform.html** and leave it open for the next set of steps. Then view the file in the browser; it should now look like Figure 11-26.

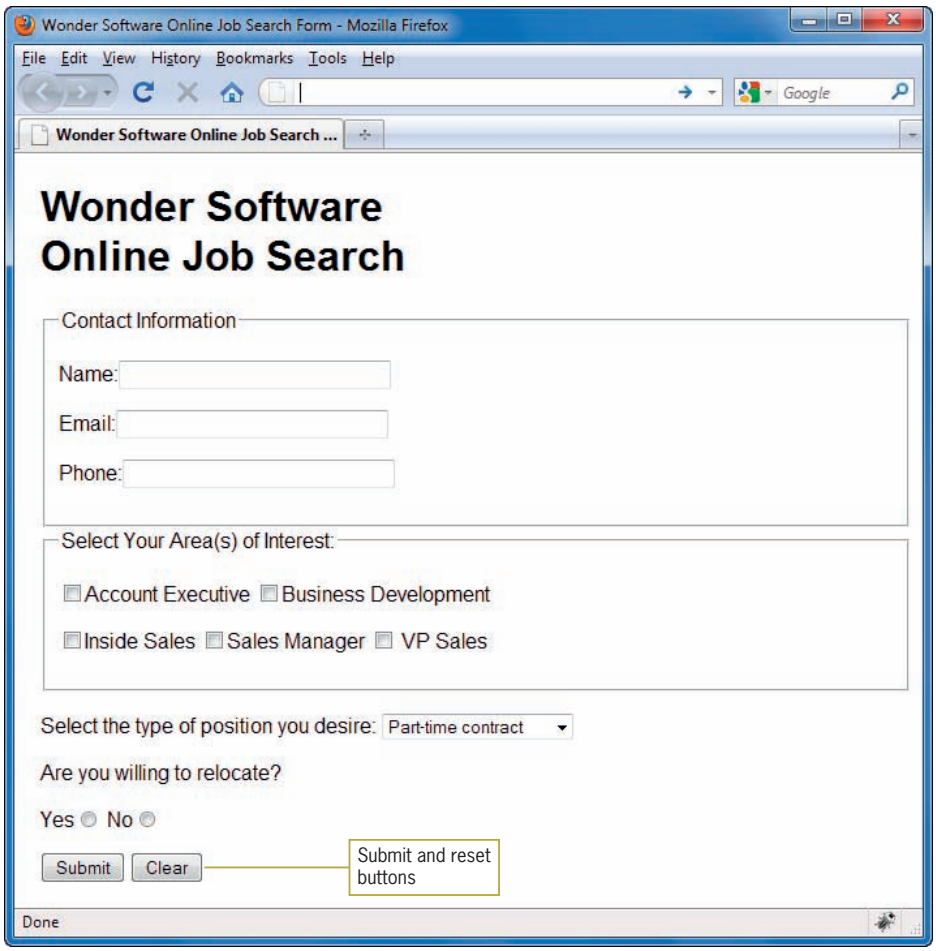


Figure 11-26 Submit and reset buttons

Styling the Labels

You will create two label styles, one for the Contact Information labels and one for the Areas of Interest labels.

To style the labels:

1. Continue working in the file **wonderform.html**.
2. Create a style for contact labels using the class name *label*. Set the properties as shown to create floating labels with right-aligned text and a margin to offset the label from the input element.

```
label.contact {
   width: 5em;
   float: left;
   text-align: right;
   margin-right: 10px;
}
```

3. Apply the style by adding the class attribute to the contact labels.

```
<fieldset>
<legend>Contact Information</legend>
<p><label for="name" class="contact">Name:</label>
   <input size="30" name="name" id="name"></p>
<p><label for="email" class="contact">Email:</label>
   <input size="30" name="email" id="email"></p>
<p><label for="phone" class="contact">Phone:</label>
   <input size="30" name="phone" id="phone"></p>
</fieldset>
```

4. Create a style for the area labels using the class name *area*. Set the properties as shown to add left and right margins to adjust the spacing of the labels for legibility.

```
label.area {
   margin-left: 2px;
   margin-right: 10px;
}
```

5. Apply the style by adding the class attribute to the area labels.

```
<fieldset>
<legend>Select Your Area(s) of Interest:</legend>


<p><input type="checkbox" name="jobtitle" value="ae"
   id="ae" /><label for="ae" class="area">Account
   Executive</label>

<input type="checkbox" name="jobtitle" value="bd"
   id="bd" /><label for="bd" class="area">Business
   Development</label></p>

<p><input type="checkbox" name="jobtitle" value="is"
   id="is" /><label for="is" class="area">Inside
   Sales</label>

<input type="checkbox" name="jobtitle" value="sm" id="sm" />
   <label for="sm" class="area">Sales Manager</label>

<input type="checkbox" name="jobtitle" value="vp" id="vp">
   <label for="vp" class="area">VP Sales</label></p>

</fieldset>
```

6. Save **wonderform.html** and leave it open for the next set of steps. Then view the file in the browser; it should now look like Figure 11-27.

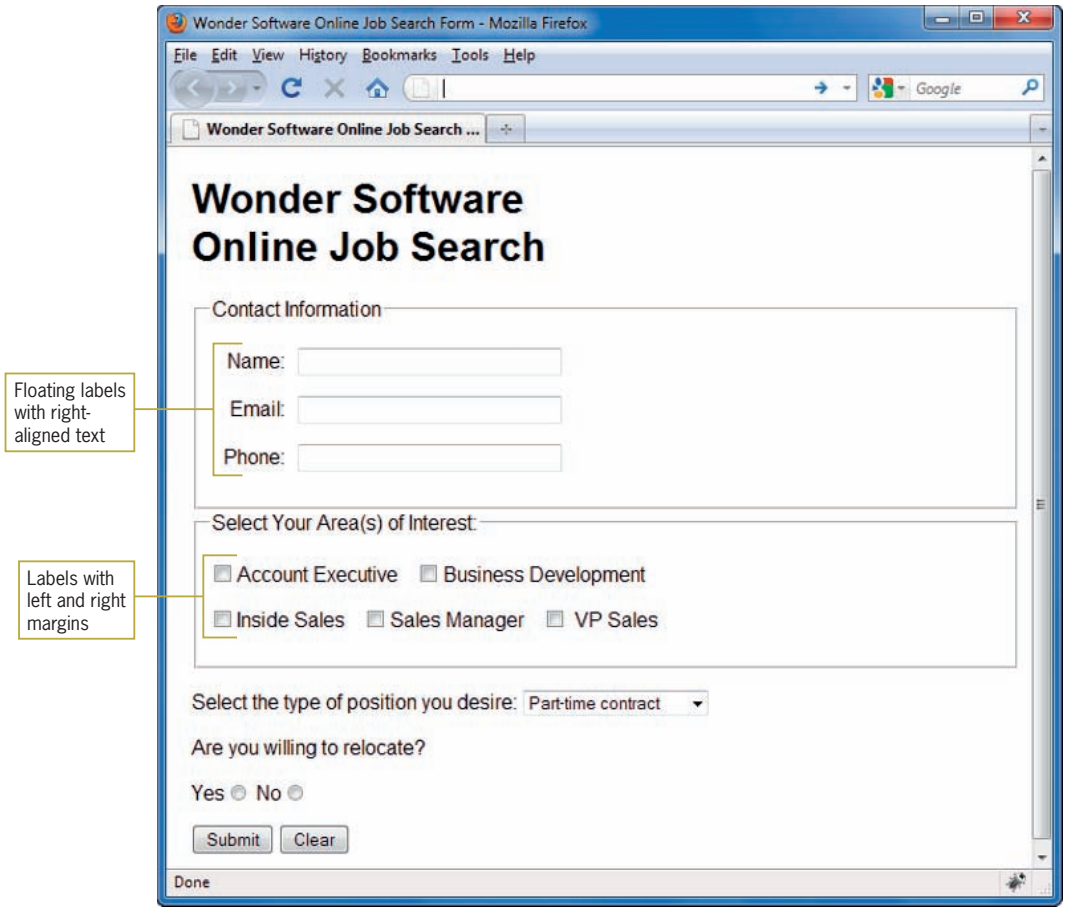


Figure 11-27 Label styles

Styling the Fieldsets and Legends

Finish the form design by styling the fieldsets and the legends.

To style the fieldsets and the legends:

1. Continue working in the file **wonderform.html.**
2. Create a style that selects both fieldsets. Set the width to 24em.

```
fieldset {
   width: 24em;
}
```

3. Provide some white space between the two fieldsets by writing a rule that selects only the area fieldset and applying a top margin of 20px. Use a class name *area* for this rule.

```
fieldset.area {
   margin-top: 20px;
}
```

4. Apply this rule to the area fieldset as shown.

```
<fieldset class="area">
<legend>Select Your Area(s) of Interest:</legend>
<p><input type="checkbox" name="jobtitle" value="ae"
   id="ae" /><label for="ae">Account Executive</label>
<input type="checkbox" name="jobtitle" value="bd" id="bd" />
   <label for="bd">Business Development</label></p>
<p><input type="checkbox" name="jobtitle" value="is"
   id="is" /><label for="is">Inside Sales</label>
<input type="checkbox" name="jobtitle" value="sm"
   id="sm" /><label for="sm">Sales Manager</label>
<input type="checkbox" name="jobtitle" value="vp"
   id="vp"> <label for="vp">VP Sales</label></p>
</fieldset>
```

5. Create a style for the legends that will apply to the <legend> element in both fieldsets.

```
legend {
   font-weight: bold;
}
```

6. Save **wonderform.html.** Then view the file in the browser; it should now look like Figure 11-28.

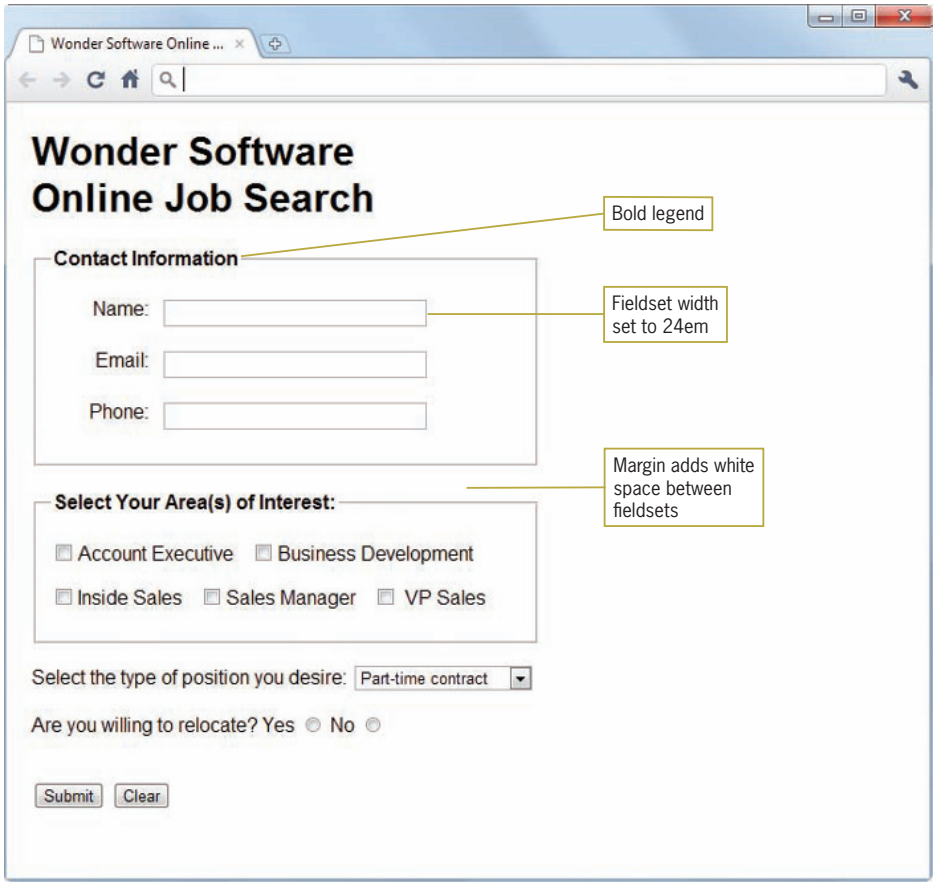


Figure 11-28 Completed form

Chapter Summary

A usable form interface is the result of choosing the correct form elements for the type of data you are requesting and designing a clear and readable form. Keep the following points in mind:

- You need to work with some type of server-based software program to process the data from your form. An HTML form is the interface for the user to enter data, and the data processing is performed on the server using applications called scripts that usually reside in the Common Gateway Interface (CGI).

- The <form> element is the container for creating a form. A form has a number of attributes that describe how the form data is handled, such as *action*, which often specifies the URL of a script file to process the form data.
- You have a variety of form elements to choose from when building a form. The <input> element defines many of the form input object types. Use the correct type of input object for the type of data you are gathering. For example, use check boxes for multiple-choice questions. For a long list of choices, use a select list.
- The <fieldset> and <legend> elements let you create more visually appealing forms that have logical groupings of <input> elements with a title.
- Most forms should be formatted to improve their legibility. The most basic formatting elements are <br/> and <p>, which place form elements on separate lines and add white space. You can avoid the ragged look of forms by using CSS to align form elements, style labels, legends, and fieldsets, or add background colors or graphics.

Key Terms

AJAX—A group of technologies that is used on the client to retrieve data from the user and submit the form request in the background.

Common Gateway Interface (CGI)—A communications bridge between the Internet and the server used as the traditional method of processing forms input. Using programs called scripts, CGI can collect data sent by a user via the Hypertext Transfer Protocol (HTTP) and transfer it to a variety of data-processing programs including spreadsheets, databases, or software running on the server.

form control—An input element in an HTML form, such as a radio button, text box, or check box.

JavaScript—A client-side scripting language used with HTML forms.

script—A program that transfers form data to a server.

Review Questions

1. What are the five commonly supported form elements?
2. What does the action attribute in the <form> element contain?
3. What are the two possible values of the form method attribute?
4. How can you group multiple check boxes together?
5. How are radio buttons different from check boxes?
6. How do you control the length of a user's entry in a text <input> element?
7. How do you enter default text in a text <input> element?
8. How do you force a check box to be selected by default?
9. What button must be included with every form?
10. How do you change the default button image for the submit button?
11. What input type lets the user attach a file to the form data?
12. What is the security problem with the password input type?
13. What are the two types of select lists?
14. What attributes let you specify the width and height of the <textarea> element?

Hands-On Projects

1. In this project, you will build text box form elements.
   a. In your HTML editor, open the file **blankform.html** in the Chapter11 folder in your work folder.
   b. Save the file as **textbox.html** in the same location.

c. Examine the code. The file contains only the default HTML elements and an empty <form> element.

d. Build the form shown in Figure 11-29. Refer to the following table for each form element's attribute values.

Name	Size	Maxlength
Street	20	35
City	20	35
State	2	35
Zip	10	35

Table 11-3 Attribute Values for the Text Box Form

e. Use CSS style properties to align the labels as shown in Figure 11-29.

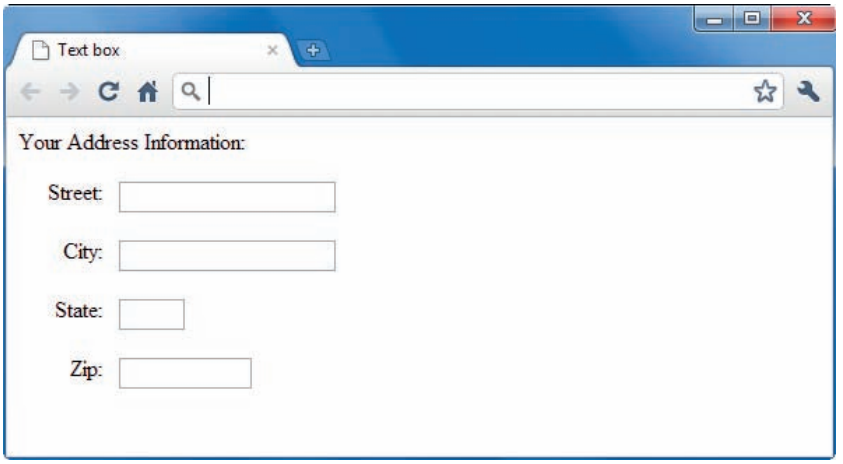


Figure 11-29 Form with text boxes

2. In this project, you will build check box form elements.

a. In your HTML editor, open the file **blankform.html** in the Chapter11 folder in your work folder.

b. Save the file as **checkbox.html** in the same location.

c. Examine the code. The file contains only the default HTML elements and an empty <form> element.

d. Build the form shown in Figure 11-30.

e. Group the check boxes with a name attribute set to "flavor."

f. Add labels that are associated with each input check box.

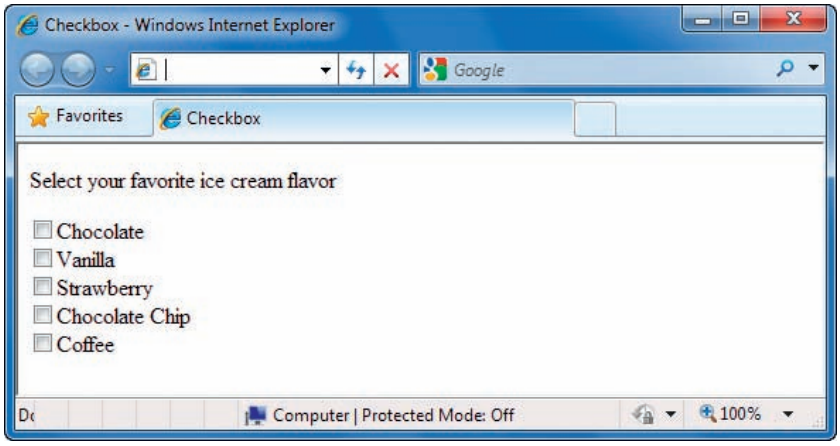


Figure 11-30 Form with check boxes

3. In this project, you build radio button form elements.

a. In your HTML editor, open the file **blankform.html** in the Chapter11 folder in your work folder.

b. Save the file as **radio.html** in the same location.

c. Examine the code. The file contains only the default HTML elements and an empty <form> element.

d. Build the form shown in Figure 11-31.

e. Make sure that "Yes" is the selected option.

f. Group the radio buttons with a name attribute set to "offer."

g. Add labels that are associated with each input radio button.

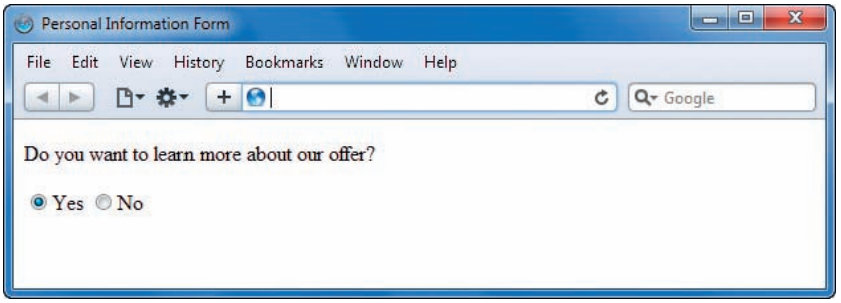


Figure 11-31 Form with radio buttons

4. In this project, you will build a text area form element.
   a. In your HTML editor, open the file **blankform.html** in the Chapter11 folder in your work folder.
   b. Save the file as **textarea.html** in the same location.
   c. Examine the code. The file contains only the default HTML elements and an empty <form> element.
   d. Build the form shown in Figure 11-32. The text area is 6 rows by 35 columns.

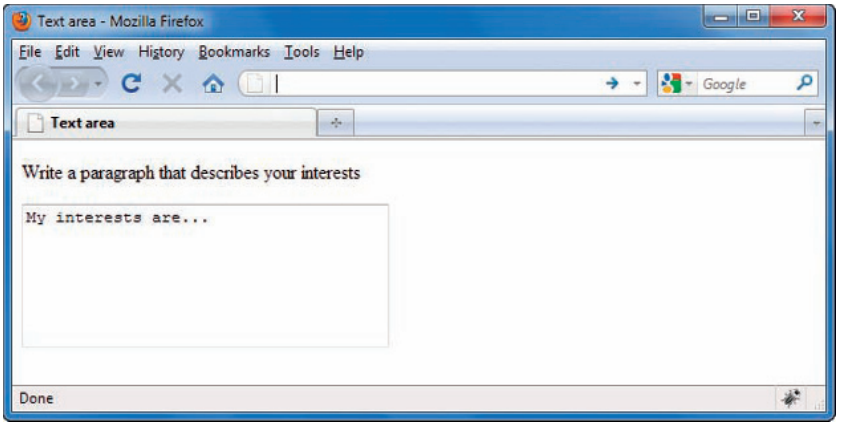


Figure 11-32 Form with text area

5. In this project, you will build a select form element.
   a. In your HTML editor, open the file **blankform.html** in the Chapter11 folder in your work folder.
   b. Save the file as **select.html** in the same location.
   c. Examine the code. The file contains only the default HTML elements and an empty <form> element.

d. Build the form shown in Figure 11-33.

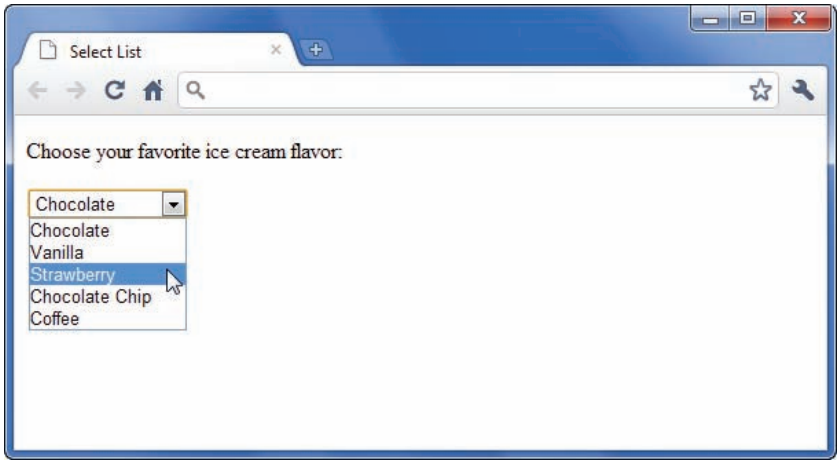


Figure 11-33 Form with list box

Individual Case Project

Build a user feedback form for your project Web site. You can refer to the sample feedback form in Chapter 3 for ideas. Customize the types of questions you ask to match the content of your site. Create both scaled questions and open-ended questions for your users. For example, ask users to rate the navigation of your site on a scale of 1 to 5, and also include a text area input where they write about their experience of navigating your Web site. Although you will not be able to activate the form (because you don't have an appropriate script to process the data), you can demonstrate the types of questions you would ask users to find out more about their habits when they visit your site.

Team Case Project

Each team member creates and submits to the instructor his or her own feedback form, as described in the preceding Individual Case Project. You are free to design the form any way you choose, but it must include the navigation characteristics, typographic specifications, and design conventions of your project Web site.

Then meet as a team and choose the best features, questions, and design characteristics from each member's submitted form. Create a new form that combines these features, and add it to your project Web site.

CHAPTER 12

Web Page Design Studio

When you complete this chapter, you will be able to:

- Apply the design, layout, and CSS skills you learned throughout this book by building a home page for a fictional Web site
- Describe the design process and decisions a Web designer must make in a standards-based development process
- Test your work in multiple browsers for consistency as you progress

In this chapter, you have a chance to apply a variety of skills as you build a mock-up layout design for a fictional Web site. Your "clients" want a home page design for their Web site named DogWorld. The site presents an online periodical of information for dog owners. The content types include feature articles, regular columnists, editorials, advertising, and a search function. The clients have employed a graphic designer to create page graphics for the site. The designer has worked from preliminary sketches provided by the clients to provide a conceptual main page design in two different color schemes. As the Web site designer, you will work with the graphic artist, testing content and browser compatibility, to implement a completed HTML version of the conceptual site.

The Initial Design Process

The clients sketched out a basic idea for the site, shown in Figure 12-1. They also submitted a Web site specification document to the graphic designer providing background information about the audience, content, and design of the Web site.

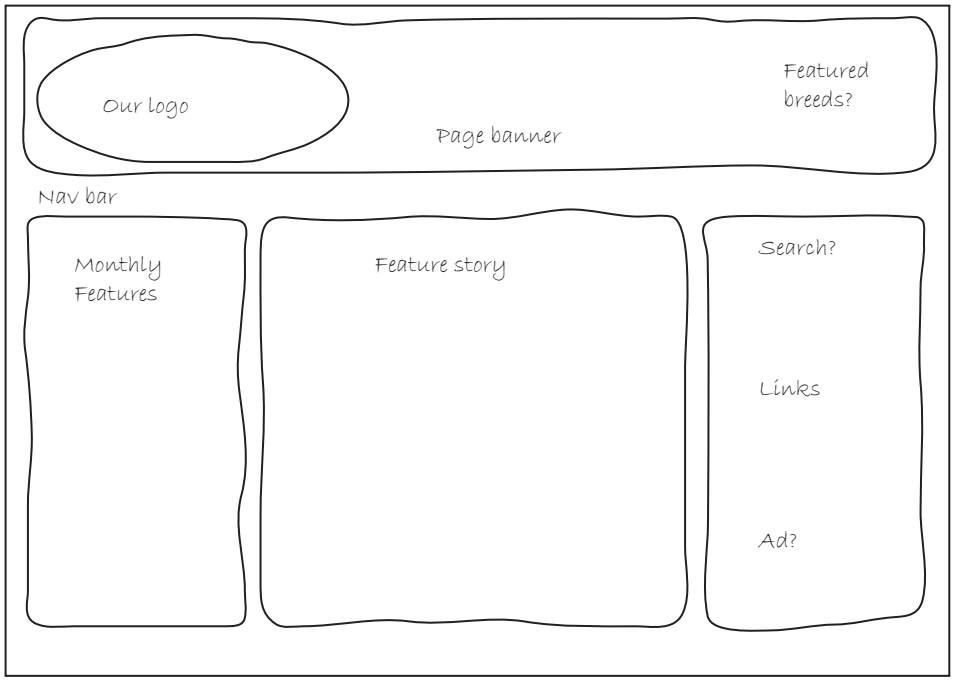


Figure 12-1 Original design sketch

Original Design Specification

- **Site title:** DogWorld
- **Content focus:** DogWorld is a special interest Web site for dog owners and fanciers. The site presents periodical-type information for dog owners in an online format. The variety of content includes feature articles, regular columnists, editorials, advertising, and a searchable archive of past articles.
- **Mission statement:** The goal of DogWorld is to inform people of the latest in news, information, trends, and products in the world of dogs. The site will have an open, accessible design that will allow ease of navigation and access to content. A search feature will let users access articles and content from the publication's entire history, using DogWorld's archive database.
- **Main elements outline**:
  - **Home page**

    A portal to the site's content, containing feature article content, links to regular columns and external Web sites, advertising, and a search feature
  - **Editorial**

    Our editor's column for sounding off on the latest dog-related issues
  - **Dog Advice**

    A Q&A between DogWorld readers and guest personalities, including trainers, vets, and breeders
  - **Reader Emails**

    A place for reader correspondence and editorial answers
  - **Training Tips**

    Basic dog training tips and techniques
  - **Breed List**

    Breed resources including links to DogWorld articles, external Web sites, a list of breeders, and breed-specific rescue shelters
- **Target audience:** The target audience for this Web site is anyone who owns or loves dogs. This is a broad audience, and the Web site must be accessible to users with a wide variety of computers, connection speeds, and computing skills.
- **Design considerations:** The Web site must be accessible in all popular browsers. Based on the latest research, our average prospective visitor uses a Windows-based PC with Internet Explorer or Firefox as their primary browser.

Based on discussions with the clients and the basic information provided in the sketch, the artist created an initial design and sent it back to the clients for approval. After the client approved the sketches, the designers created the wireframe shown in Figure 12-2. This wireframe shows a more complete version of the page designs, including navigation elements, search functions, advertising space, and the like. The wireframe gives the client and the team more information to validate the page design.

For more on wireframes, see Chapter 3.

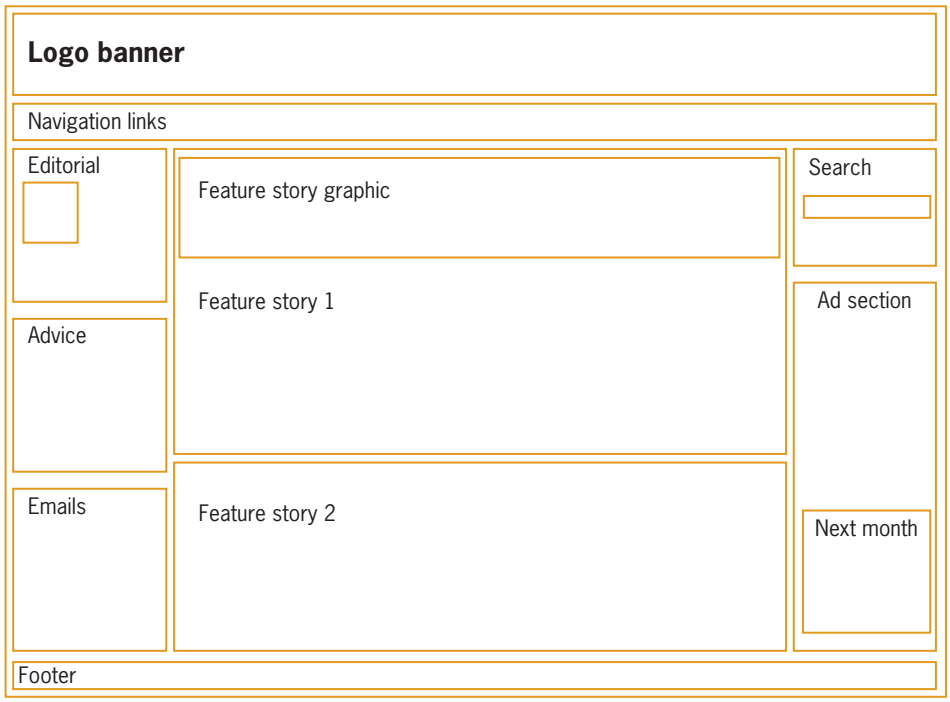


Figure 12-2 Designer's wireframe design

After the wireframe was approved, the designers can start drafting page mockups that show color, text, and more layout details. Figure 12-3 shows the designer's first draft of the Web site. The designer created this initial design mockup using Adobe Photoshop, so it is not an actual Web page, but a picture of what the finished Web site will look like.

Figure 12-3 Designer's first draft of the page design

The clients reviewed and critiqued the design. Although they liked the basic layout, they did have some specific complaints. They thought the logo area was too plain and the page banner contained too many pictures of dogs. The text overall was too small, and the feature boxes in the left column needed more white space and seemed cluttered. The clients also wanted the dog bone image to look like a dog tag and decided to remove the highlighting on the link text in the right column.

In addition, the clients suggested a dog-related shape in the background of the page banner to make it more interesting, and wanted to see the entire page in different color schemes. The clients also noted that the brown text in the feature image was hard to read.

The designer went back to the drawing board and returned with two new versions of the site, shown in Figures 12-4 and 12-5. This new page design incorporated many of the changes the clients requested. The red version of the color scheme gave the clients a chance to explore something other than the aqua and brown scheme the designer suggested. After reviewing both of the new versions, the clients decided to move forward with the aqua and brown design shown in Figure 12-4.

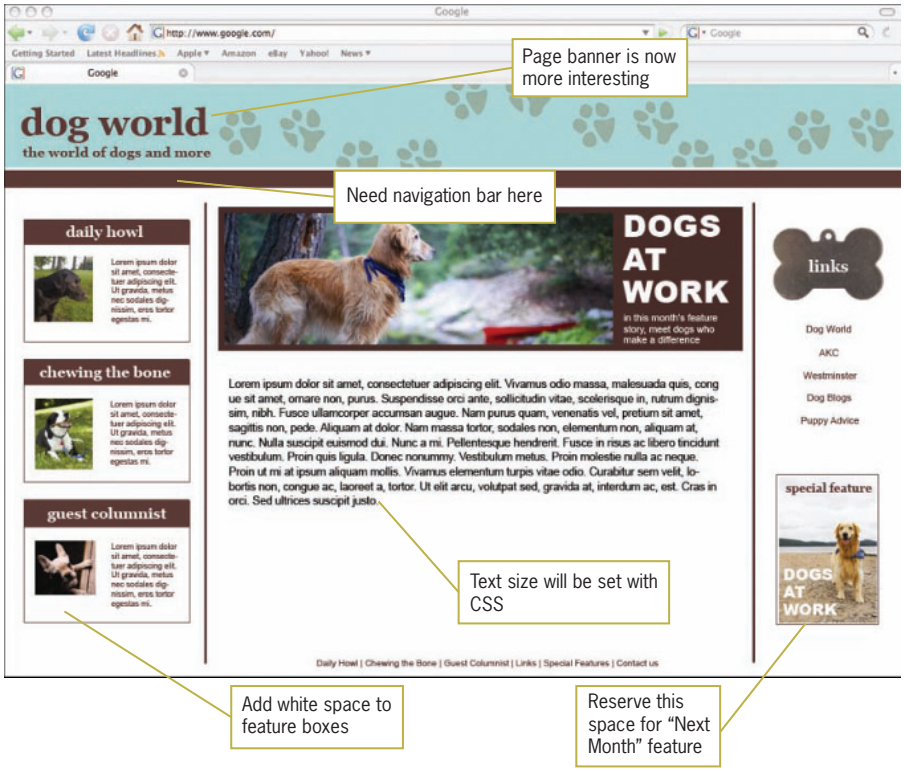


Figure 12-4 Second draft of the site in aqua and brown

Figure 12-5 Second draft of the site in red and brown

The clients knew they wanted to make other changes, including adding a search form and a space to advertise upcoming articles. They returned the draft of the Web site to the designer for one more set of changes, including adding a navigation bar, a search form, and space for a planned second feature article, as shown in Figure 12-6.

Figure 12-6 Final draft of the Web page design

After seeing the graphic designer's final draft, the clients were happy enough with the conceptual art to hand it over to the Web designer, who will build some mock-up pages for further testing and refinement. This is your project for the chapter.

Creating the Mock-up Web Page

In the following set of steps, you build a fixed page layout version of the design that the clients approved. The designer provided you with the images and color information you need to re-create the design using HTML.

Figure 12-7 shows the results you need to achieve and columns that create the layout for the page. The page is a fixed design based on a 1024 × 768 screen resolution. The layout uses floating division elements to create the three-column layout. The column borders are highlighted with lines in a contrasting color that will not appear in the finished Web page. This figure also shows the

measurement values you will use in the CSS code to create the wrapper and column elements.

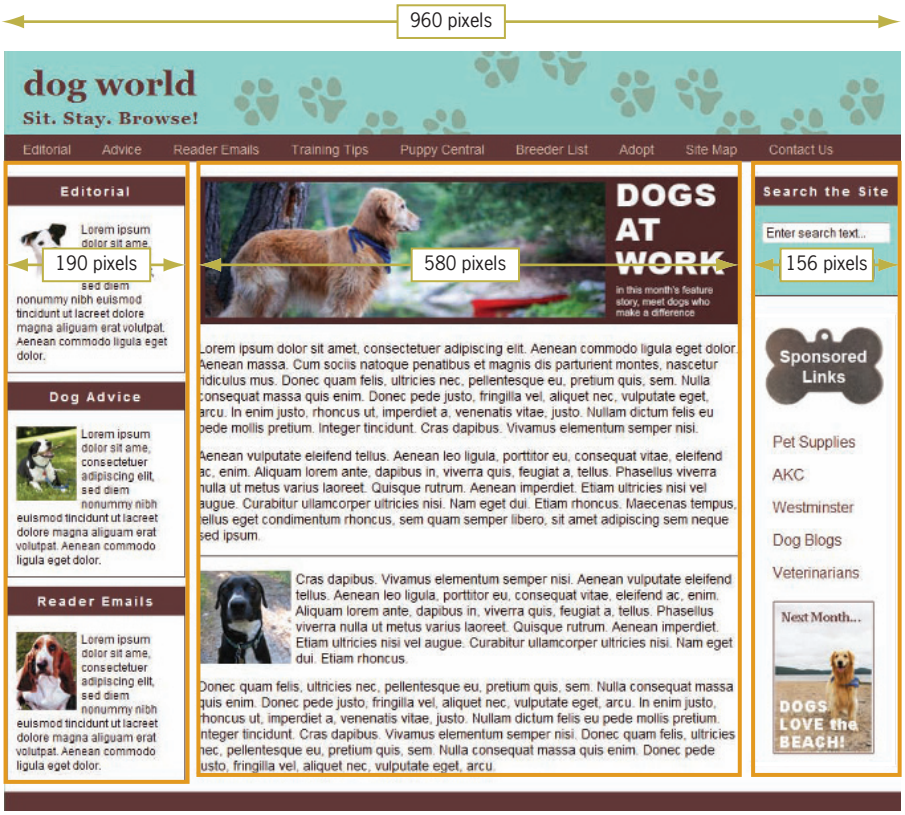


Figure 12-7 Home page design showing page layout and column measurements

You need to determine the widths of the columns so you know the pixel measurements to use in the design. The graphic designer provided the Dogs at Work image as 580 pixels wide. This image sets the width for the middle content column. If the content wrapper width is 960 pixels, deducting the width of the middle column leaves 380 pixels (960 – 580 = 380).

Now deduct the planned width of the two outside columns. The left column is 190 pixels, so 380 – 190 = 190. Deduct the width of the right column, 190 – 156 = 34. This remaining amount of

space will be used to create the gutters between the columns, which will be 15 pixels on both sides of the main column, plus 2 pixels of space on the outside margin of both the left and right columns. Figure 12-8 shows the gutter measurements in the layout.

Figure 12-8 Gutter measurements in the layout

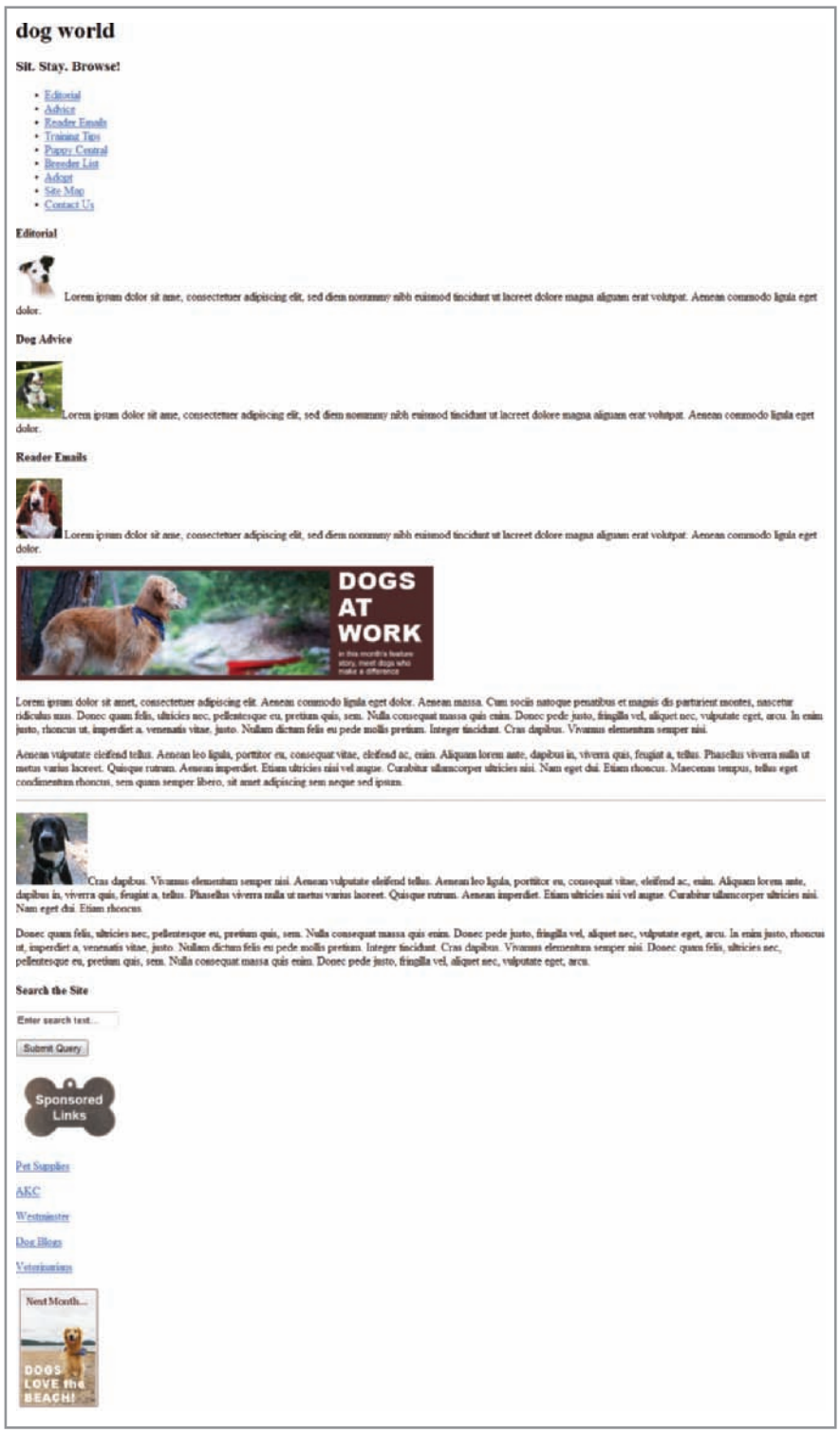
The graphic designer provided you with a variety of images from the initial design. You can use these when you create the mock-up of the layout. Figure 12-9 shows the images and their filenames.

Figure 12-9 Image files used in the home page design

Examining the HTML Code

To start working on the file:

1. Open the file **dogworld.html** in your HTML editor, and save it in your work folder as **dogworld1.html**.
2. Copy the following image files into your work folder.
   - dogs_at_work.jpg
   - smfeature1.jpg
   - smfeature2.jpg
   - smfeature3.jpg
   - featureimg2.jpg
   - links_img.jpg
   - headerbkg.jpg
   - next_month.jpg
3. Open the file **dogworld1.html** in your browser. When you open the file, it looks like Figure 12-10. Notice that this is a basic, left-aligned Web page, with all the images and text in place.



dog world

Sit. Stay. Browse!

- Editorial
- Advice
- Reader Emails
- Training Tips
- Puppy Central
- Breeder List
- Adopt
- Site Map
- Contact Us

Editorial

Lorem ipsum dolor sit ame, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat. Aenean commodo ligula eget dolor.

Dog Advice

Lorem ipsum dolor sit ame, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat. Aenean commodo ligula eget dolor.

Reader Emails

Lorem ipsum dolor sit ame, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat. Aenean commodo ligula eget dolor.

DOGS AT WORK

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi.

Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum.

Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus.

Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu.

Search the Site

Enter search text...

Submit Query

Sponsored Links

Pet Supplies

AKC

Westminster

Dog Blogs

Veterinarians

Next Month...

DOGS LOVE the BEACH!

Figure 12-10 Original HTML document with no CSS

4. View the source code in your browser or text editor. The file contains the standard HTML elements including an empty <style> element in the head section where you will add the style rules. Comments in the code (colored blue) indicate where content will reside in the finished layout, such as in the Header or Left Column. The complete code for the page follows:

```
<!DOCTYPE html>

<html>
<head>
<title>Welcome to Dog World!</title>
<meta content="text/html; charset=utf-8"
   http-equiv="Content-Type" />
<style type="text/css">

/*
Brown: #613838
Red: #e32026
Green: #93d4cd
*/

</style>
</head>

<body>

<!-- Header -->
<h1>
dog world
</h1>
<h3>
Sit. Stay. Browse!
</h3>

<!-- Navigation -->
<ul>
<li><a href=" ">Editorial</a></li>
<li><a href=" ">Advice</a></li>
<li><a href=" ">Reader Emails</a></li>
<li><a href=" ">Training Tips</a></li>
<li><a href=" ">Puppy Central</a></li>
<li><a href=" ">Breeder List</a></li>
<li><a href=" ">Adopt</a></li>
<li><a href=" ">Site Map</a></li>
<li><a href=" ">Contact Us</a></li>
</ul>


<!-- Left Column -->

<h4>Editorial</h4>
<p>
```

```
<img src="smfeature1.jpg" width="64" height="63"
alt="Editorial Image" border="0" />
placeholder text</p>

<h4>Dog Advice</h4>
<p>
<img src="smfeature2.jpg" width="64" height="79"
alt="Dog Advice Image" border="0" />
placeholder text</p>

<h4>Reader Emails</h4>
<p>
<img src="smfeature3.jpg" width="64" height="83"
alt="Reader Emails Image" border="0"  />
placeholder text</p>

<!-- Main Column -->

<img src="dogs_at_work.jpg" width="580" height="158"
alt="Dogs at Work Image" title="Dogs at Work Feature
   Story" />
<p>placeholder text</p>

<p>placeholder text</p>

<hr/>

<p><img src="featureimg2.jpg" width="100"
alt="Dog photo" border="0" />placeholder text</p>

<p>placeholder text</p>
</div>

<!-- Right Column -->

<h4>Search the Site</h4>
<form method="get" action=" ">
<p><input type="text" value="Enter search text..."
maxlength="25" size="18" /></p>
<p><input type="submit" /></p>
</form>

<img src="links_img.jpg" width="148" height="106"
alt="Sponsored Links" title="Sponsored Links"
   border="0" />
<p><a href=" ">Pet Supplies</a></p>
<p><a href=" ">AKC</a></p>
<p><a href=" ">Westminster</a></p>
<p><a href=" ">Dog Blogs</a></p>
<p><a href=" ">Veterinarians</a></p>
<img src="next_month.jpg" width="121" height="176"
alt="Next Month Image" border="0" />

</body>
</html>
```

The finished code for the completed page design is included for your reference at the end of the chapter.

Setting Page Background Color and Default Font

Start building the page layout by specifying the default font and the background color for the page.

1. Continue working in the file dogworld.html.
2. Locate the style elements in the head section of the document. Add a comment after the opening <style> tag as shown. As the style sheet grows in complexity, these comments will let you easily locate a particular section of the style sheet.

```
<style type="text/css">

/* ------------ Page Background ------------ */
```

3. Write a style rule that selects the body element. Set the font-family to arial with a fallback value of sans-serif.

Refer to Chapter 5 for more information on font-family fallback values.

```
/* ------------ Page Background ------------ */


body {
   font-family: arial, sans-serif;
}
```

4. Add the background-color property, and set the value to green (#8cad9a).

```
body {
   font-family: arial, sans-serif;
   background-color: #8cad9a;
}
```

Creating the Page Wrapper Division

To create the page wrapper division:

1. Continue working in the file dogworld.html.
2. Locate the <style> element in the head section of the document. Add a CSS comment to indicate where the page wrapper style rule will reside.

```
/* ------------ Page Wrapper  ------------ */
```

3. Write the style rule for the wrapper division that will contain the page content. Set the width to 960 pixels and the background color to white (#fff).

```
/* ------------ Page Wrapper  ------------ */

div#wrapper {
   width: 960px;
   background-color: #fff;
}
```

4. Because this is a fixed page design, you will want the page to auto-center regardless of the user's browser size or screen resolution. Also, you want to add a border so you can see the result in the browser. You will remove this border later.

Refer to Chapter 7 for more information on page wrappers and layout designs.

```
/* ------------ Page Wrapper  ------------ */

div#wrapper {
   width: 960px;
   background-color: #fff;
   margin-left: auto;
   margin-right: auto;
   border: solid thin black;
}
```

5. Now apply the wrapper division to the document. Immediately after the opening <body> tag, add the <division> element with wrapper id as shown.

```
<body>
<div id="wrapper"><!-- Open Wrapper -->
```

6. Close this division by adding a tag just before the closing </body> tag. Add the HTML comment as shown to mark this tag as the close of the wrapper container.

```
</div> <!-- Close Wrapper -->
</body>
</html>
```

7. Save your work and view it in your browser. Figure 12-11 shows the results. The wrapper division, indicated by the black border, contains all of the page content, and is automatically centered in the browser window.

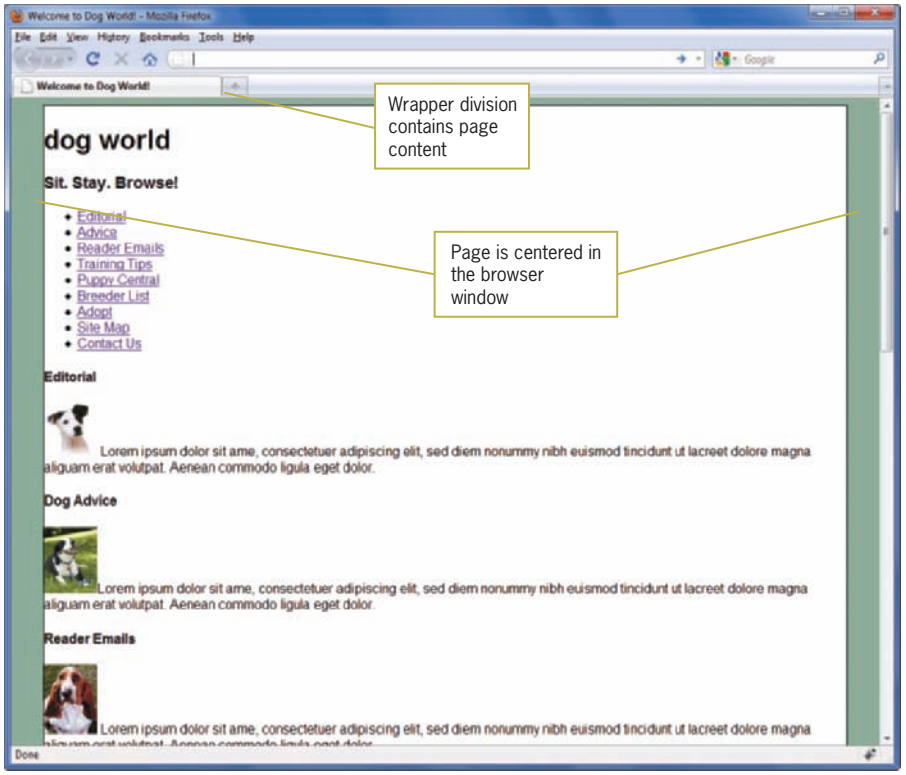


Figure 12-11 Wrapper division contains all of the page content

8. Remove the border property indicated in the following code from the wrapper style rule, then save the file and continue to the next procedure.

```
div#wrapper {
   width: 960px;
   background-color: #fff;
   margin-left: auto;
   margin-right: auto;
   border: solid thin black;
}
```

Remember to test your layout in multiple browsers as you progress through the design process.

Creating the Page Header

To create the page header section:

1. Add a CSS comment to indicate where the page header style rules will reside.

```
/* ------------ Page Header  ------------ */
```

2. Write a style rule that selects the division element with an id named *header*. This division will contain a background graphic that is 960 pixels wide by 95 pixels high, which you add later. You will set an explicit width and height because you know the exact size of the content that will reside within this division.

```
div#header {
   width: 960px;
   height: 95px;
}
```

3. Add the background image to the header.

```
div#header {
   width: 960px;
   height: 95px;
   background-image: url(headerbkg.jpg);
}
```

Refer to Chapter 8 for more information on background images.

4. Locate the comment that identifies the header section, and add the opening and closing tags as shown. Make sure to set the id to *header*.

```
<!-- Header -->
<div id="header">
<h1>
   dog world
</h1>
<h3>
   Sit. Stay. Browse!
</h3>
</div>
```

5. Save the file and view it in your browser. Figure 12-12 shows the results. The header division contains the background image you specified. Notice the white margin above the header section. This is the <h1> element's top margin. You will style the header text later in the chapter.

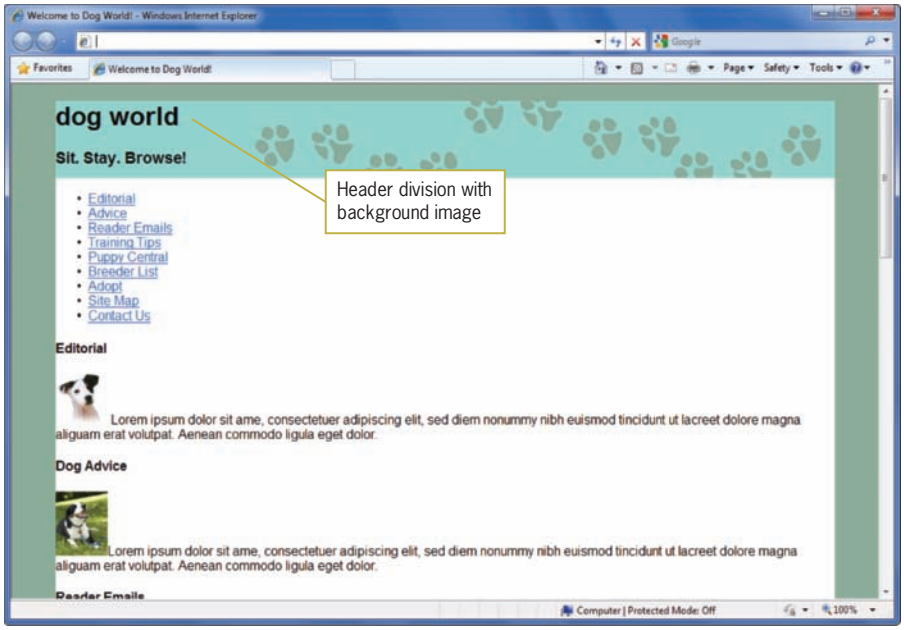


Figure 12-12 Header division with background image

Styling the Header Text

Now that you have the background image in place, you can style the header text.

To style the header text:

1. Locate the header text, which is contained in an <h1> element for the head text "dog world" and an <h3> element for the subheading "Sit. Stay. Browse!".
2. Add a class attribute with the value *head* to the <h1> element.
3. Add a class attribute with the value *subhead* to the <h3> element. Your header division should now look like the following code:

```
<!-- Header -->
<div id="header">
<h1 class="head">
   dog world
</h1>
<h3 class="subhead">
   Sit. Stay. Browse!
</h3>
</div>
```

4. Add a comment to the style area for the header text.

```
/* ------------ Header Text ------------ */
```

5. Write a rule to set shared properties for both headings. The selector specifies both element and their class names.

```
/* ------------ Header Text ------------ */

h1.head, h3.subhead { }
```

6. Write a style rule that sets the font to Georgia with a fallback value of serif, a left margin of 20 pixels, and a color of brown (#613838).

```
/* ------------ Header Text ------------ */

h1.head, h3.subhead {
   font-family: georgia, serif;
   margin-left: 20px;
   color: #613838;
}
```

7. Write a separate style rule for the head text that sets the font-size to 2.25em and adds 15 pixels of padding to the top of the text.

```
/* ------------ Header Text ------------ */

h1.head, h3.subhead {
   font-family: georgia, serif;
   margin-left: 20px;
   color: #613838;
}

h1.head {
   font-size: 2.25em;
   padding-top: 15px;
}
```

8. Write a separate style rule for the subhead text that sets the top margin to a negative value of 20 pixels to place the text closely together. Add letter-spacing of 1 pixel between each letter in the subhead.

```
/* ------------ Header Text ------------ */

h1.head, h3.subhead {
   font-family: georgia, serif;
   margin-left: 20px;
   color: #613838;
}
```

```
h1.head {
   font-size: 2.25em;
   padding-top: 15px;
}

h3.subhead {
   margin-top: -20px;
   letter-spacing: 1px;
}
```

9. Save the file and view it in your browser. Figure 12-13 shows the results. The header division contains the styled text you specified.

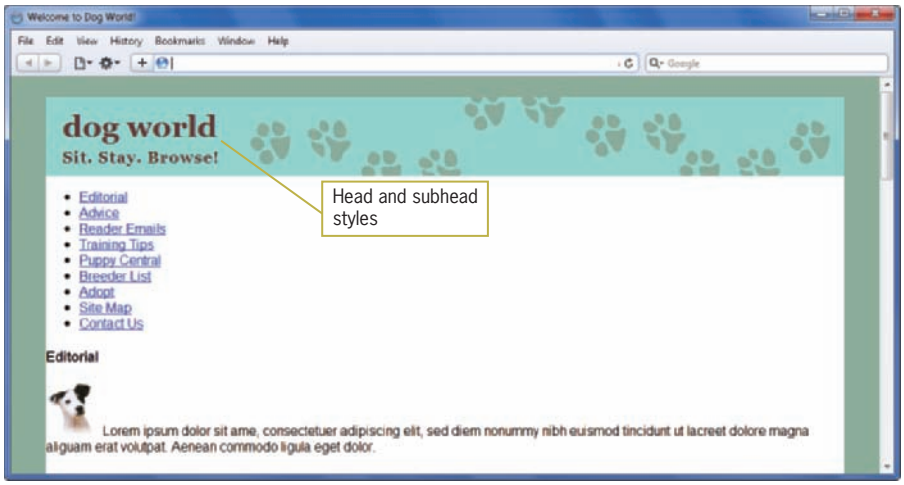


Figure 12-13 Completed header text styles

Creating the Navigation Bar

The navigation bar is an unordered list element that contains the list of navigation links for the Web site. There are three steps to create the navigation bar:

- Style the list item elements to display them horizontally.
- Style the list container to add a background color and white space.
- Style the anchor elements to change the color and add a hover interaction.

Styling the List Elements

To style the list elements:

1. Locate the navigation list in the HTML code. Add the id *navlist* to the opening <ul> tag as shown.

```
<!-- Navigation -->
<ul id="navlist">
<li><a href=" ">Editorial</a></li>
<li><a href=" ">Advice</a></li>
<li><a href=" ">Reader Emails</a></li>
<li><a href=" ">Training Tips</a></li>
<li><a href=" ">Puppy Central</a></li>
<li><a href=" ">Breeder List</a></li>
<li><a href=" ">Adopt</a></li>
<li><a href=" ">Site Map</a></li>
<li><a href=" ">Contact Us</a></li>
</ul>
```

2. In the style area, add a CSS comment to indicate where the page header style rules will reside.

```
/* ------------ Navigation Bar------------ */
```

3. Write a style rule that selects the <li> elements within the unordered list using the id *navlist*.

```
/* ------------ Navigation Bar------------ */

ul#navlist li { }
```

4. Set the display property to *inline* and the list-style-type to *none* to remove the default bullets.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
}
```

5. Add 30 pixels of padding to the right side of every <li> element to provide space between the links in the navigation bar.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}
```

6. Save the file and view it in your browser. Figure 12-14 shows the results. The list items are displayed horizontally with the bullets removed. Notice that the list wraps to the next line. You will fix this when you style the <ul> element next.

Figure 12-14 List elements displayed horizontally

Styling the List Container

To style the list container:

1. Write a new rule that selects the <ul> element with the id *navlist.*

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist { }
```

2. Add a property to set the background-color to brown (#613838).

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
}
```

3. Save the file and view it in your browser. Figure 12-15 shows the results. You need to remove the default white space above the navigation bar that is built in to the <ul> element.

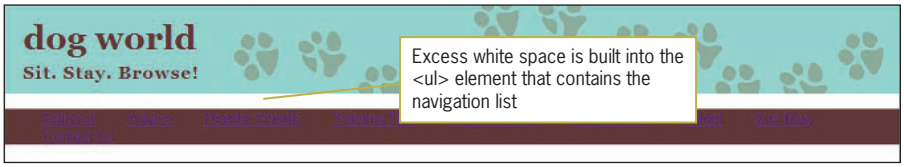


Figure 12-15 Navigation bar with excess white space

4. Add the margin-top property set to a negative value of 5 pixels to remove the excess space between the header and the navigation bar.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
   margin-top: -5px;
}
```

5. Save the file and view it in your browser. Figure 12-16 shows how the extra white space has been removed, leaving the navigation bar flush against the header division. Notice the links text is not visible against the brown background. You will fix this in the next set of steps.

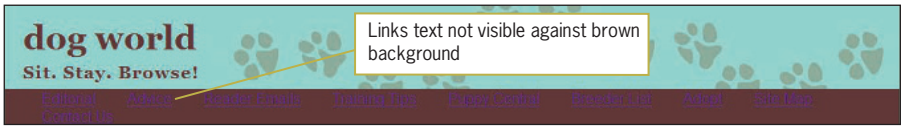


Figure 12-16 Navigation bar with excess white space removed

Styling the Navigation Anchor Elements

To style the navigation anchor elements

1. Write a new rule that selects the <a> element contained within the <li> elements in the *navlist.*

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}
```

```
ul#navlist {
   background-color: #613838;
   margin-top: -5px;
}

ul#navlist li a { }
```

2. Change the color of the links to light brown (#c6b29e) so they can be seen against the brown background. Set the text-decoration to *none* to remove default underlining.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
   margin-top: -5px;
}

ul#navlist li a {
   color:#c6b29e;
   text-decoration: none;
}
```

3. Save the file and view it in your browser. Figure 12-17 shows the new link colors. Notice the links still wrap to the next line and need some changes in spacing. You will fix this in the next steps.

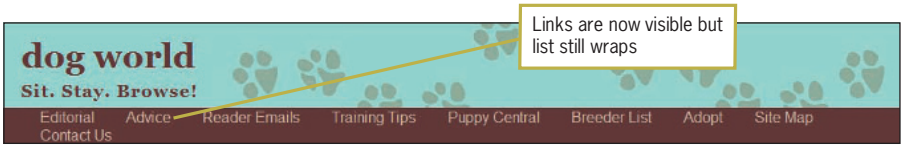


Figure 12-17 New link colors in the navigation bar

4. Change the font size of the links to make them smaller by adding the font-size property and setting it to .85em.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
```

```
   background-color: #613838;
   margin-top: -5px;
}

ul#navlist li a {
   color:#c6b29e;
   text-decoration: none;
   font-size: .85em;
}
```

5. Now go back to the selector that specifies the <ul> element and add padding to increase the white space. This rule adds 5 pixels of padding to the top, right, and bottom sides, and 20 pixels to the left side.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
   margin-top: -5px;
   padding: 5px 5px 5px 20px;
}

ul#navlist li a {
   color:#c6b29e;
   text-decoration: none;
   font-size: .85em;
}
```

6. Finally, write a rule that selects the <a> elements and adds the hover pseudo-class to change the link color when the user points to each link.

```
/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
   margin-top: -5px;
   padding: 5px 5px 5px 20px;
}
```

```
ul#navlist li a {
   color:#c6b29e;
   text-decoration: none;
   font-size: .85em;
}

ul#navlist li a:hover {color: #fff;}
```

7. Save the file and view it in your browser. Figure 12-18 shows the completed navigation bar with padding and the hover pseudo-class.

Figure 12-18 Completed navigation bar with hover effect and increased padding

Creating the Column Divisions

You will now create the column divisions to contain the page contents. Figure 12-19 shows the three columns you will create and their id names. The column widths and margins were determined by the designer.

- *leftcol*—190 pixels wide, 2-pixel left margin
- *maincol*—580 pixels wide, 15-pixel left and right margins
- *rightcol*—156 pixels wide, 2-pixel right margin

Figure 12-19 Column divisions and id names

Creating the Left Column

The left column contains the small feature boxes and floats to the left edge of the layout.

To create the left content column:

1. In the style area, add a CSS comment to indicate where the page column style rules will reside.

```
/* ------------ Page Columns------------ */
```

2. Write a new style rule that selects the division with the id *leftcol*.

```
/* ------------ Page Columns------------ */

div#leftcol { }
```

3. Add the column properties. This column is 190 pixels wide and floats to the left of the layout. The left margin value adds a few pixels to the default border built into the browser, separating the left column from the left edge of the browser window.

```
/* ------------ Page Columns------------ */

div#leftcol {
   float: left;
   width: 190px;
   margin-left: 2px;
}
```

4. Locate the comment that identifies the left column division, and add the opening and closing tags as shown. Make sure to set the id to *leftcol*.

```
<!-- Left Column -->

<div id="leftcol">
<h4>Editorial</h4>
<p>
<img src="smfeature1.jpg" width="64" height="63"
   alt="Editorial Image" border="0" class="left" />
Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.</p>

<h4>Dog Advice</h4>
<p>
<img src="smfeature2.jpg" width="64" height="79"
   alt="Dog Advice Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.
</p>

<h4>Reader Emails</h4>
<p>
<img src="smfeature3.jpg" width="64" height="83"
   alt="Reader Emails Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.
</p>
</div>
```

5. Save the file and view it in your browser. Figure 12-20 shows the left column in place, indicated by the orange border in the illustration. Currently the column flows outside of the wrapper because the left column is the

only floating element. You will solve this problem as you continue building the layout.

Figure 12-20 Left content column

Creating the Main Column

The main column contains the feature and secondary articles and two images. The main column floats left to align with the left column.

To create the main content column:

1. Write a new style rule that selects the division with the id *maincol.*

```
/* ------------ Page Columns------------ */

div#leftcol {
   float: left;
   width: 190px;
   margin-left: 2px;
}

div#maincol { }
```

2. Add the column properties. This column is 580 pixels wide and floats to the left. Specify right and left margins of 15 pixels each.

```
div#maincol {
   float: left;
   width: 580px;
   margin-left: 15px;
   margin-right: 15px;
}
```

3. Locate the comment that identifies the main column division, and add the opening and closing tags as shown. Make sure to set the id to *maincol*.

```
<!-- Main Column -->

<div id="maincol">
<img src="dogs_at_work.jpg" width="580" height="158"
   alt="Dogs at Work Image" title="Dogs at Work Feature
   Story" />
<p Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Aenean commodo ligula eget dolor. Aenean massa.
Cum sociis natoque penatibus et magnis dis parturient
montes, nascetur ridiculus mus. Donec quam felis,
ultricies nec, pellentesque eu, pretium quis, sem.
Nulla consequat massa quis enim. Donec pede justo,
fringilla vel, aliquet nec, vulputate eget, arcu. In
enim justo, rhoncus ut, imperdiet a, venenatis vitae,
justo. Nullam dictum felis eu pede mollis pretium.
Integer tincidunt. Cras dapibus. Vivamus elementum
semper nisi. </p>

<p>Aenean vulputate eleifend tellus. Aenean leo ligula,
porttitor eu, consequat vitae, eleifend ac, enim.
Aliquam lorem ante, dapibus in, viverra quis, feugiat a,
tellus. Phasellus viverra nulla ut metus varius laoreet.
Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi
vel augue. Curabitur ullamcorper ultricies nisi. Nam
eget dui. Etiam rhoncus. Maecenas tempus, tellus eget
condimentum rhoncus, sem quam semper libero, sit amet
adipiscing sem neque sed ipsum. </p>

<hr/>

<p><img src="featureimg2.jpg" width="100" alt="Dog
photo" border="0"/>Cras dapibus. Vivamus elementum
semper nisi. Aenean vulputate eleifend tellus. Aenean
leo ligula, porttitor eu, consequat vitae, eleifend ac,
enim. Aliquam lorem ante, dapibus in, viverra quis,
feugiat a, tellus. Phasellus viverra nulla ut metus
varius laoreet. Quisque rutrum. Aenean imperdiet.
Etiam ultricies nisi vel augue. Curabitur ullamcorper
ultricies nisi. Nam eget dui. Etiam rhoncus. </p>
```

```
<p>Donec quam felis, ultricies nec, pellentesque eu,
pretium quis, sem. Nulla consequat massa quis enim.
Donec pede justo, fringilla vel, aliquet nec, vulputate
eget, arcu. In enim justo, rhoncus ut, imperdiet a,
venenatis vitae, justo. Nullam dictum felis eu pede
mollis pretium. Integer tincidunt. Cras dapibus.
Vivamus elementum semper nisi. Donec quam felis,
ultricies nec, pellentesque eu, pretium quis, sem.
Nulla consequat massa quis enim. Donec pede justo,
fringilla vel, aliquet nec, vulputate eget, arcu.</p>
</div>
```

4. Save the file and view it in your browser. Figure 12-21 shows the main column in place, indicated by the border drawn in a contrasting color in the illustration. The layout is starting to look more organized as the columns are completed.

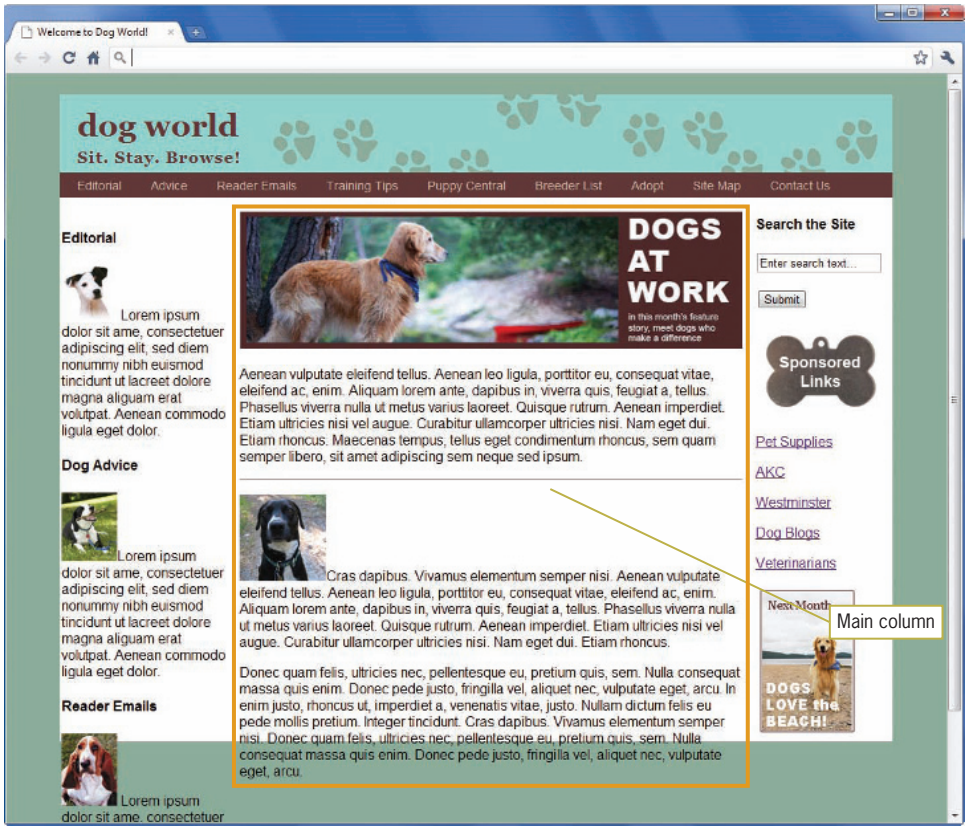


Figure 12-21 Main content column

Creating the Right Column

The right column contains the search form, sponsored links to other Web sites, and the Next Month feature.

To create the right content column:

1. Write a new style rule that selects the division with the id *rightcol*.

```
/* ------------ Page Columns------------ */

div#leftcol {
   float: left;
   width: 190px;
   margin-left: 2px;
}

div#maincol {
   float: left;
   width: 580px;
   margin-left: 15px;
   margin-right: 15px;
}

div#rightcol { }
```

2. Add the column properties. This column is 156 pixels wide and floats to the left. Specify a 2-pixel right margin.

```
div#rightcol {
   width: 156px;
   float: left;
   margin-right: 2px;
}
```

3. Locate the comment that identifies the right column division, and add the opening and closing tags as shown. Make sure to set the id to *rightcol*.

```
<!-- Right Column -->

<div id="rightcol">
<h4>Search the Site</h4>
<form method="get" action=" ">
<p class="form"><input type="text" value="Enter search
   text..." maxlength="25" size="18" /></p>
<p class="form"><input type="submit" /></p>
</form>
```

```
<img src="links_img.jpg" class="links" width="148"
   height="106" alt="Sponsored Links" title="Sponsored
   Links Section" border="0" />
<p><a href=" ">Pet Supplies</a></p>
<p><a href=" ">AKC</a></p>
<p><a href=" ">Westminster</a></p>
<p><a href=" ">Dog Blogs</a></p>
<p><a href=" ">Veterinarians</a></p>
<img class="nextmonth" src="next_month.jpg" width="121"
   height="176" alt="Next Month Image" border="0" />
</div>
```

4. Save the file and view it in your browser. Figure 12-22 shows the wrapper that created the white background for the page seems to have disappeared. This is because the floats are not being contained properly, as described in Chapter 7. You can solve this problem by adding a footer to the layout in the next set of steps.

Figure 12-22 Floated columns are not contained correctly

Creating the Footer

Adding a nonfloating footer element (in the normal flow), with the clear property set to *both*, will force the containing wrapper to extend to contain all elements.

To add the footer:

1. Scroll down to the bottom of the file, and locate the wrapper's closing tag indicated by the comment you added earlier.

```
</div> <!-- Close Wrapper -->
</body>
</html>
```

2. Add a comment to identify the footer section and a div element with an id *footer* as shown. The footer contains no text content; it will only be used to correct the float problem shown in Figure 12-22 and add a colored border to the bottom of the layout.

```
<!-- Footer -->
<div id="footer">

</div>

</div> <!-- Close Wrapper -->
</body>
</html>
```

3. In the style area, add a rule in the Page Columns area that selects the footer division.

```
div#footer { }
```

4. Add properties to set the background color to brown (#613838) and the height to 20px.

```
div#footer {
   background-color: #613838;
   height: 20px;
   }
```

5. Add the clear property set to *both* to fix the float problem.

```
div#footer {
   background-color: #613838;
   height: 20px;
   clear: both;
   }
```

6. Save the file and view it in your browser. Figure 12-23 shows the completed columns contained properly within the wrapper.

Figure 12-23 Three columns and footer properly contained within the wrapper

Building the Small Feature Boxes

The left column of the DogWorld Web page contains three small feature boxes that highlight the content of the monthly columns, as shown in Figure 12-24. Each feature box contains a text-reverse heading, a photo, and some copy. Three feature boxes are stacked in the left column. You will use elements to create the feature boxes.

Figure 12-24 Small feature boxes

To build the small feature boxes:

1. Locate the left column division. Three <h4> elements, each with accompanying text and an image, make up the small feature elements.

   Add a element that contains each set of content, one each for the "Editorial," "Dog Advice," and "Reader Emails" sections as shown in the following code. Make sure to set the id to *smfeature*.

```
<!-- Left Column -->

<div id="leftcol">

<div class="smfeature">
<h4>Editorial</h4>
```

```
<p>
<img src="smfeature1.jpg" width="64" height="63"
   alt="Editorial Image" border="0" class="left" />
Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.</p>
</div>

<div class="smfeature">
<h4>Dog Advice</h4>
<p>
<img src="smfeature2.jpg" width="64" height="79"
   alt="Dog Advice Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.</p>
</div>

<div class="smfeature">
<h4>Reader Emails</h4>
<p>
<img src="smfeature3.jpg" width="64" height="83"
   alt="Reader Emails Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
adipiscing elit, sed diem nonummy nibh euismod
   tincidunt ut lacreet dolore magna aliguam erat
   volutpat. Aenean commodo ligula eget dolor.</p>
</div>


</div>
```

2. In the style area, add a CSS comment to indicate where the small feature boxes style rules will reside.

```
/* ------------ Small Feature Boxes ------------ */
```

3. Write a style rule with *smfeature* as the class name. You will use a class selector rather than an id selector because there is more than one small feature division.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature { }
```

4. Set the width to 190 pixels and the height to auto. Add a thin solid brown border (#613838) and a bottom margin of 10 pixels to separate the boxes in the left column.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
```

```
    border: thin solid #613838;
    margin-bottom: 10px;
}
```

5. Save the file and view it in the browser. Figure 12-25 shows the result of the style rule. The smfeature division is 190 pixels wide, with the specified brown border. You will style the <h4> heading text next.

Figure 12-25 Basic smfeature division—190 pixels wide, brown border

6. Write a new style rule for the smfeature h4 class. Use descendant selection to make sure this style rule applies only to <h4> elements when they appear within a smfeature division as shown.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
    width: 190px;
    height: auto;
    border: thin solid #613838;
    margin-bottom: 10px;
}

div.smfeature h4 { }
```

7. Write the style rules for the <h4> heading text. Set the font-family to arial, the font-size to 85%, the text color to white (#fff), and the background color to brown (#613838).

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}

div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
}
```

8. Save your work and view it in the browser. Figure 12-26 shows the result of the style rule. The <h4> heading text is now white text reversed against the brown background. Notice the white space above the heading that must be removed.

Figure 12-26 Heading text

9. Finish styling the <h4> heading text. Align the text to center, and add 2 pixels of letter spacing and .5 em of padding. Add the margin-top property set to 0 pixels to remove the element's default white space.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
```

```
    border: thin solid #613838;
    margin-bottom: 10px;
}

div.smfeature h4 {
    font-family: arial, sans-serif;
    font-size: 85%;
    color: #fff;
    background: #613838;
    text-align: center;
    letter-spacing: 2px;
    padding: .5em;
    margin-top: 0px;
}
```

10. Save your work and view it in the browser. Figure 12-27 shows the result of adding the final three properties to the h4 heading text style rule.

Figure 12-27 Completed h4 heading text

Styling the Small Feature Box Content

Now that you have created the basic structure of the small feature box, you can style the text and images they contain.

1. Write a new style rule to set style properties for the <p> elements that reside within the small feature box. Use a descendant selector to apply the style only to the <p> elements that are children of the smfeature division.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}
div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
   text-align: center;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}

div.smfeature p { }
```

2. Set the font-family to arial with a fallback value of sans-serif, and set the font-size to 75%. Set the text color to black, and add a margin of 10 pixels to add some white space between the paragraph text and the border of the smfeature box that contains it.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}

div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
   text-align: center;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}

div.smfeature p {
   font-family: arial, sans-serif;
   font-size: 75%;
   color: #000;
   margin: 10px;
}
```

Remember that it's easy to test different font sizes. If you're not happy with the font size, change the value in the style rule, save, and reload the page in the browser.

3. Add a new rule that selects the <img> within the smfeature division.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}

div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
   text-align: center;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}

div.smfeature p {
   font-family: arial, sans-serif;
   font-size: 75%;
   color: #000;
   margin: 10px;
}

div.smfeature img { }
```

4. Use the float property to float the image to the left of the paragraph text. Add a 5-pixel right margin to add white space between the image and the text.

```
/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}

div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
   text-align: center;
```

```
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}

div.smfeature p {
   font-family: arial, sans-serif;
   font-size: 75%;
   color: #000;
   margin: 10px;
}

div.smfeature img { }
   float: left;
   margin-right: 5px;
}
```

5. Save the file and view the results in your browser. Figure 12-28 shows the completed small feature boxes in the left column.

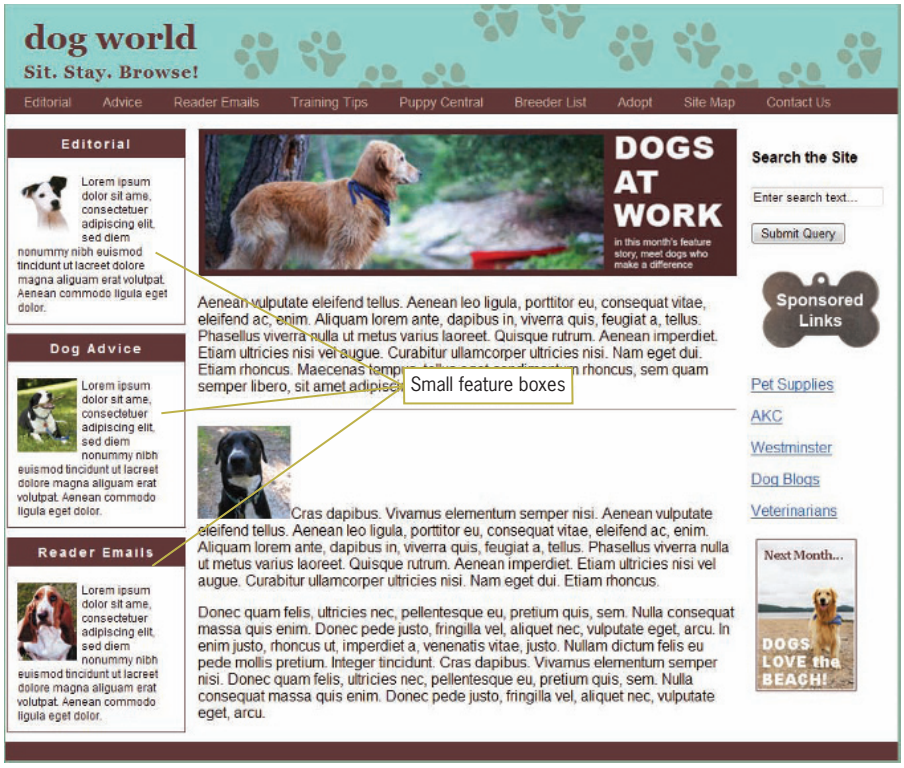


Figure 12-28 Completed small feature boxes

Building the Feature Article Section

The middle column of the layout contains the feature story image, two sections of text separated by a dividing rule, and a secondary image, as shown in Figure 12-29.

Figure 12-29 Feature article section

To build the feature article section:

1. In the style area, add a CSS comment to indicate where the feature article style rules will reside.

```
/* ------------ Feature Article ------------*/
```

2. Write a new style rule that selects the <p> elements that reside within the maincol division.

```
/* ------------ Feature Article ------------*/

div#maincol p { }
```

3. Set the font size to 85% and the line height to 1.25em to increase the legibility of the article text.

```
/* ------------ Feature Article ------------*/

div#maincol p {
   font-size: 85%;
   line-height: 1.25em;
}
```

4. Save the file and view the results in your browser. Figure 12-30 shows the new text size and increased line height. Notice the image in the lower section needs to be floated so the article text will wrap around it. You will fix this next.

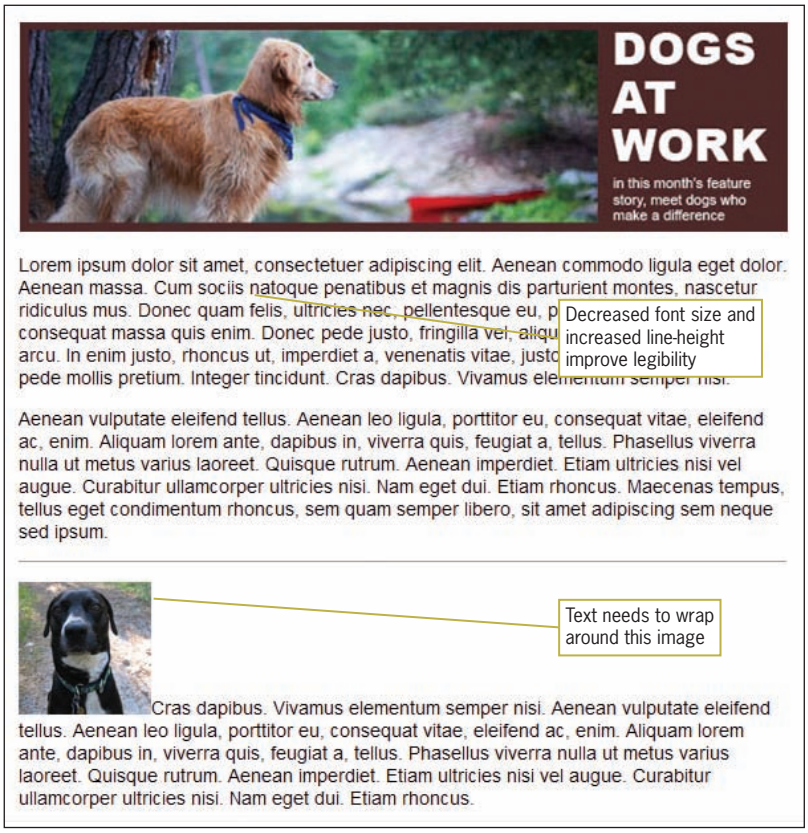


Figure 12-30 Completed feature article text style

5. Locate the second image in the maincol section in the paragraph that follows the <hr /> element. Add a class attribute with a value *feature2* to the <img> element as shown.

```
<hr/>

<p><img src="featureimg2.jpg" width="100" alt="Dog
photo" border="0" class="feature2" />Cras
dapibus. Vivamus elementum semper nisi. Aenean
vulputate eleifend tellus. Aenean leo ligula, porttitor
eu, consequat vitae, eleifend ac, enim. Aliquam lorem
ante, dapibus in, viverra quis, feugiat a, tellus.
Phasellus viverra nulla ut metus varius laoreet.
Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi
vel augue. Curabitur ullamcorper ultricies nisi. Nam
eget dui. Etiam rhoncus. </p>
```

6. Write a new style rule that selects the <img> element with the class *figure2*.

```
/* ------------ Feature Article ------------*/

div#maincol p {
   font-size: 85%;
   line-height: 1.25em;
}

img.feature2 { }
```

7. Add style properties to float the image to the left and add a 5-pixel right margin.

```
/* ------------ Feature Article ------------*/

div#maincol p {
   font-size: 85%;
   line-height: 1.25em;
}

img.feature2 {
   float: left;
   margin-right: 5px;
}
```

8. Save the file and view the results in your browser. Figure 12-31 shows the completed feature article section.

Figure 12-31 Completed feature article section

Styling the Search and Links Section

The right column of the layout contains the search form, links section, and two images: the links dog tag and the "Next Month" feature graphic, as shown in Figure 12-32. The search form lets users search the articles archive for specific content. You will build this section first.

Figure 12-32 Search and links section

To build the search section:

1. Locate the search form located in the right column. Add a element to contain the form with an id of *searchbox*.

```
<div id="searchbox">
<h4>Search the Site</h4>
<form method="get" action=" ">
<p class="form"><input type="text" value="Enter search
   text..." maxlength="25" size="18" /></p>
<p class="form"><input type="submit" /></p>
</form>
</div>
```

2. In the style area, add a CSS comment to indicate where the search box style rules will reside.

```
/* ------------ Search Box ------------ */
```

3. Write a style rule that selects a division with *searchbox* as the id name.

```
div#searchbox { }
```

4. Set the width to 155 pixels and the height to auto. Add a thin solid brown border (#613838) and a bottom margin of 20 pixels to separate the search box from the links section below it in the right column.

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 20px;
}
```

5. Add a background color green (#93d4cd) that matches the layout colors and align the text to the center of the column.

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 20px;
   background: #93d4cd;
   text-align: center;
}
```

6. Save the file and view it in the browser. Figure 12-33 shows the partially completed search box.

Figure 12-33 Partially completed search box

7. Write a new style rule for the searchbox h4 class. Use descendant selection to make sure this style rule applies to the <h4> element that resides within the searchbox division as shown.

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 20px;
   background: #93d4cd;
   text-align: center;
}

div#searchbox h4 { }
```

8. Write the style rules for the <h4> heading text. Set the font-size to 85%, the text color to white (#fff), and the background color to brown (#613838).

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 20px;
   background: #93d4cd;
   text-align: center;
}

div#searchbox h4 {
   font-size: 85%;
   color: #fff;
   background-color: #613838;
}
```

9. Finish styling the <h4> heading text. Add 2 pixels of letter spacing and .5em of padding. Add the margin-top property set to 0 pixels to remove the element's default white space, as you did previously with the small feature boxes.

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 20px;
   background: #93d4cd;
   text-align: center;
}

div#searchbox h4 {
   font-size: 85%;
   color: #fff;
   background-color: #613838;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}
```

10. Save the file and view the results in your browser. Figure 12-34 shows the completed search box section.

Figure 12-34 Completed search box

Building the Links Section

The links section contains two images and a variety of hypertext links contained within a brown border.

To style the links section:

1. Locate the links content located in the right column. Add a element to contain the links content with an id of *links*.

```
<div id="links">
<img src="links_img.jpg" class="links" width="148"
   height="106" alt="Sponsored Links" title="Sponsored
   Links Section" border="0" />
<p><a href=" ">Pet Supplies</a></p>
<p><a href=" ">AKC</a></p>
<p><a href=" ">Westminster</a></p>
<p><a href=" ">Dog Blogs</a></p>
<p><a href=" ">Veterinarians</a></p>
<img class="nextmonth" src="next_month.jpg" width="121"
   height="176" alt="Next Month Image" border="0" />
</div>
```

2. In the style sheet, add a CSS comment to indicate where the links style rules will reside.

```
/* ------------ Links Section  ------------ */
```

3. Write the style rule for the links division you just created. Set the width of the division to 150 pixels and the height to auto, and add a solid thin brown (#613838) border.

```
/* ------------ Links Section  ------------ */

div#links {
   width: 150px;
   height: auto;
   border: solid thin #613838
}
```

4. Save the file and view the results in your browser. Figure 12-35 shows the partially completed links section. The link colors need to match the Web site colors, and both the links and bottom image need to be aligned properly. You will do this next.

Figure 12-35 Partially completed links section

5. Write a new style rule that selects the <p> elements within the links division that contain the hypertext link text.

```
/* ------------ Links Section  ------------ */

div#links {
   width: 150px;
   height: auto;
   border: solid thin #613838;
}

div#links p { }
```

6. Add a left padding value of 20 pixels to the link text paragraphs.

```
div#links p {padding-left: 20px;}
```

7. Style the link colors by writing two selectors for the <a> elements: one for the link pseudo-class and one for the visited pseudo-class. Set the text color for each to brown (#613838).

```
/* ------------ Links Section  ------------ */
div#links{
   width: 150px;
   height: auto;
   border: solid thin #613383;
}
div#links p {padding-left: 20px;}
div#links p a:link {color: #613838;}
div#links p a:visited {color: #613838;}
```

8. Write a final style rule for the hypertext links that uses the :hover pseudo-class to provide bottom border underlining when the user points at the link. You will also have to add a text decoration property in a separate rule to turn off the default hypertext linking.

```
/* ------------ Links Section  ------------ */
div#links{
   width: 150px;
   height: auto;
   border: solid thin #613383;
}
div#links p {padding-left: 20px;}
div#links p a:link {color: #613838;}
div#links p a:visited {color: #613838;}
div#links p a:hover {border-bottom: solid #613838;}
div#links p a {text-decoration: none;}
```

9. Save the file and view the results in your browser. Figure 12-36 shows the partially completed results of the link styling. The last style you add in the next step will align the Next Month image.

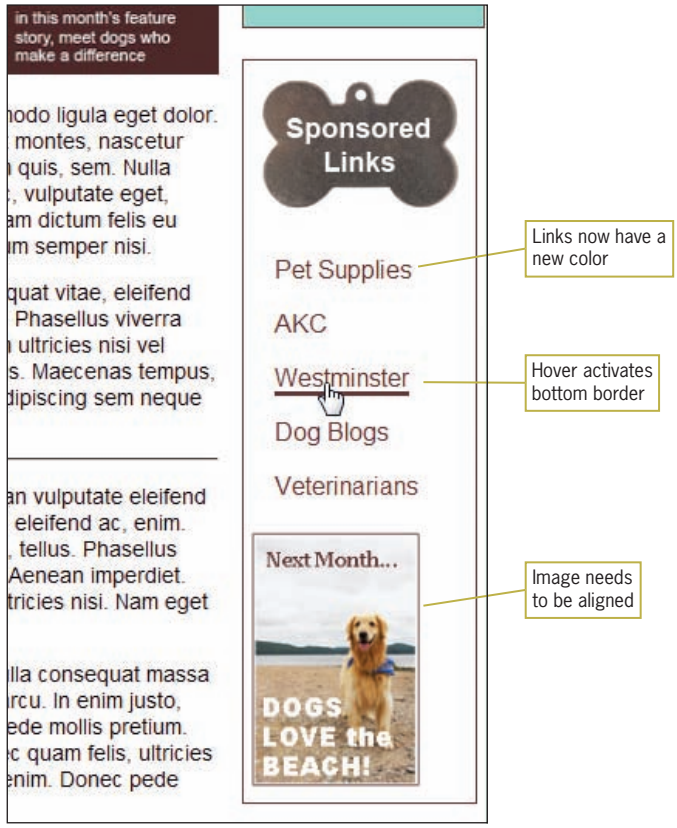


Figure 12-36 Styled hypertext links

10. Locate the Next Month image in the right column. Add a class of *nextmonth* as shown.

```
<div id="links">
<img src="links_img.jpg" width="148" height="106"
   alt="Sponsored Links" title="Sponsored Links Section"
   border="0" />
<p><a href=" ">Pet Supplies</a></p>
<p><a href=" ">AKC</a></p>
<p><a href=" ">Westminster</a></p>
<p><a href=" ">Dog Blogs</a></p>
<p><a href=" ">Veterinarians</a></p>
<img class="nextmonth" src="next_month.jpg"
   width="121" height="176" alt="Next Month Image"
   border="0" />
</div>
```

11. Write a style rule that selects the image with class *nextmonth*. Set the left padding value to 15 pixels.

```
/* ------------ Links Section  ------------ */
div#links{
   width: 150px;
   height: auto;
   border: solid thin #613383;
}

div#links p {padding-left: 20px;}
div#links p a:link {color: #613838;}
div#links p a:visited {color: #613838;}
div#links p a:hover {border-bottom: solid #613838;}
div#links p a {text-decoration: none;}

img.nextmonth {padding-left: 15px;}
```

12. Save the file and view the final Web page design results in your browser. Figure 12-37 shows the completed Web page.

Figure 12-37 Completed Web page design

You created a complete layout based on the graphic designer's original design. Although you have been testing the page in different browsers during the development process, it is a good idea to test a final time to make sure the pages are displayed properly.

Figure 12-38 Finished layout in four browsers

Figure 12-38 shows the page in a finished layout in Firefox, Chrome, Internet Explorer, and Safari, respectively. The page layout holds together well in all browsers.

Finished Page Code

Following is the complete code for the page. Notice the CSS and HTML comments that identify each section of the style sheet and the table code. Remember that your style sheet rules can appear in a different order than shown here.

```
<!DOCTYPE html>

<html>
<head>
   <title>Welcome to Dog World!</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-
   Type" />
<style type="text/css">

/*
Brown: #613838
Red: #e32026
Green: #93d4cd
*/

/* ------------ Page Background ------------ */
body {
   font-family: arial, sans-serif;
   background-color: #8cad9a;
}

/* ------------ Page Wrapper  ------------ */

div#wrapper {
   width: 960px;
   background-color: #fff;
   margin-left: auto;
   margin-right: auto;
}

/* ------------ Page Header  ------------ */

div#header {
   width: 960px;
   height: 95px;
   background-image: url(headerbkg.jpg);
}

/* ------------ Header Text ------------ */

h1.head, h3.subhead {
   font-family: georgia, serif;
   margin-left: 20px;
   color: #613838;
}
```

```
h1.head {
   font-size: 2.25em;
   padding-top: 15px;
}

h3.subhead {
   margin-top: -20px;
   letter-spacing: 1px;
}

/* ------------ Navigation Bar------------ */

ul#navlist li {
   display: inline;
   list-style-type: none;
   padding-right: 30px;
}

ul#navlist {
   background-color: #613838;
   margin-top: -5px;
   padding: 5px 5px 5px 20px;
}

ul#navlist li a {
   color:#c6b29e;
   text-decoration: none;
   font-size: .85em;
}

#navlist li a:hover {color: #fff;}

/* ------------ Page Columns------------ */

div#leftcol {
   float: left;
   width: 190px;
   margin-left: 2px;
}

div#maincol {
   float: left;
   width: 580px;
   margin-left: 15px;
   margin-right: 15px;
}

div#rightcol {
   width: 156px;
   float: left;
   margin-right: 2px;
}
```

```
div#footer {
   background-color: #613838;
   height: 20px;
   clear: both;
}

/* ------------ Small Feature Boxes ------------ */

div.smfeature {
   width: 190px;
   height: auto;
   border: thin solid #613838;
   margin-bottom: 10px;
}

div.smfeature h4 {
   font-family: arial, sans-serif;
   font-size: 85%;
   color: #fff;
   background: #613838;
   text-align: center;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;

}

div.smfeature p {
   font-family: arial, sans-serif;
   font-size: 75%;
   color: #000;
   margin: 10px;
}

div.smfeature img {
   float: left;
   margin-right: 5px;
}


/* ------------ Feature Article ------------ */

div#maincol p {
   font-size: 85%;
   line-height: 1.25em;
}


img.feature2 {
   float:left;
   margin-right: 5px;
}
```

```
/* ------------ Search Box  ------------ */
div#searchbox {
   width: 155px;
   height: auto;
   border: thin solid #613838;
   background: #93d4cd;
   margin-bottom: 20px;
   text-align: center;
}

div#searchbox h4 {
   font-size: 85%;
   color: #fff;
   background-color: #613838;
   letter-spacing: 2px;
   padding: .5em;
   margin-top: 0px;
}

/* ------------ Links Section  ------------ */
div#links{
   width: 150px;
   height: auto;
   border: solid thin #613838;
}

div#links p {padding-left: 20px;}
div#links p a:link {color: #613838;}
div#links p a:visited {color: #613838;}
div#links p a:hover {border-bottom: solid #613838;}
div#links p a {text-decoration: none;}
img.nextmonth {padding-left: 15px;}

</style>
</head>

<body>

<div id="wrapper"> <!-- Open Wrapper -->

<!-- Header -->
<div id="header">
<h1 class="head">
dog world
</h1>
<h3 class="subhead">
Sit. Stay. Browse!
</h3>
</div>

<!-- Navigation -->
```

```
<ul id="navlist">
<li><a href=" ">Editorial</a></li>
<li><a href=" ">Advice</a></li>
<li><a href=" ">Reader Emails</a></li>
<li><a href=" ">Training Tips</a></li>
<li><a href=" ">Puppy Central</a></li>
<li><a href=" ">Breeder List</a></li>
<li><a href=" ">Adopt</a></li>
<li><a href=" ">Site Map</a></li>
<li><a href=" ">Contact Us</a></li>
</ul>

<!-- Left Column -->

<div id="leftcol">

<div class="smfeature">
<h4>Editorial</h4>

<p>
<img src="smfeature1.jpg" width="64" height="63"
   alt="Editorial Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
   adipiscing elit, sed diem nonummy nibh euismod tincidunt ut
   lacreet dolore magna aliguam erat volutpat. Aenean commodo
   ligula eget dolor.</p>
</div>

<div class="smfeature">
<h4>Dog Advice</h4>
<p>
<img src="smfeature2.jpg" width="64" height="79"
   alt="Dog Advice Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
   adipiscing elit, sed diem nonummy nibh euismod tincidunt ut
   lacreet dolore magna aliguam erat volutpat. Aenean commodo
   ligula eget dolor.</p>
</div>

<div class="smfeature">
<h4>Reader Emails</h4>
<p>
<img src="smfeature3.jpg" width="64" height="83"
   alt="Reader Emails Image" border="0" class="left" />
   Lorem ipsum dolor sit ame, consectetuer
   adipiscing elit, sed diem nonummy nibh euismod tincidunt ut
   lacreet dolore magna aliguam erat volutpat. Aenean commodo
   ligula eget dolor.</p>
</div>

</div><!-- End Left Column -->

<!-- Main Column -->
```

```
<div id="maincol">
<img src="dogs_at_work.jpg" width="580" height="158" alt="Dogs
at Work Image" title="Dogs at Work Feature Story" />
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aenean commodo ligula eget dolor. Aenean massa. Cum sociis
natoque penatibus et magnis dis parturient montes, nascetur
ridiculus mus. Donec quam felis, ultricies nec, pellentesque
eu, pretium quis, sem. Nulla consequat massa quis enim. Donec
pede justo, fringilla vel, aliquet nec, vulputate eget, arcu.
In enim justo, rhoncus ut, imperdiet a, venenatis vitae,
justo. Nullam dictum felis eu pede mollis pretium. Integer
tincidunt. Cras dapibus. Vivamus elementum semper nisi. </p>

<p>Aenean vulputate eleifend tellus. Aenean leo ligula,
porttitor eu, consequat vitae, eleifend ac, enim. Aliquam
lorem ante, dapibus in, viverra quis, feugiat a, tellus.
Phasellus viverra nulla ut metus varius laoreet. Quisque
rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue.
Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam
rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem
quam semper libero, sit amet adipiscing sem neque sed ipsum.
</p>

<hr/>

<p><img src="featureimg2.jpg" width="100" alt="Dog photo"
border="0" class="feature2" />Cras dapibus. Vivamus elementum
semper nisi. Aenean vulputate eleifend tellus. Aenean leo
ligula, porttitor eu, consequat vitae, eleifend ac, enim.
Aliquam lorem ante, dapibus in, viverra quis, feugiat a,
tellus. Phasellus viverra nulla ut metus varius laoreet.
Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel
augue. Curabitur ullamcorper ultricies nisi. Nam eget dui.
Etiam rhoncus. </p>

<p>Donec quam felis, ultricies nec, pellentesque eu, pretium
quis, sem. Nulla consequat massa quis enim. Donec pede justo,
fringilla vel, aliquet nec, vulputate eget, arcu. In enim
justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam
dictum felis eu pede mollis pretium. Integer tincidunt. Cras
dapibus. Vivamus elementum semper nisi. Donec quam felis,
ultricies nec, pellentesque eu, pretium quis, sem. Nulla
consequat massa quis enim. Donec pede justo, fringilla vel,
aliquet nec, vulputate eget, arcu.</p>
</div>
<!-- End Main Column -->

<!-- Right Column -->

<div id="rightcol">
```

```
<div id="searchbox">
<h4>Search the Site</h4>
<form method="get" action=" ">
<p class="form"><input type="text" value="Enter search text..."
   maxlength="25" size="18" /></p>
<p class="form"><input type="submit" /></p>
</form>
</div>

<div id="links">
<img src="links_img.jpg" class="links" width="148"
height="106" alt="Sponsored Links" title="Sponsored Links"
border="0" />
<p><a href=" ">Pet Supplies</a></p>
<p><a href=" ">AKC</a></p>
<p><a href=" ">Westminster</a></p>
<p><a href=" ">Dog Blogs</a></p>
<p><a href=" ">Veterinarians</a></p>
<img class="nextmonth" src="next_month.jpg" width="121"
   height="176" alt="Next Month Image" border="0" />
</div>

</div>
<!-- End Right Column -->

<!-- Footer -->
<div id="footer">

</div>

<!-- End Footer -->

</div> <!-- Close Wrapper -->


</body>
</html>
```

Chapter Summary

In this chapter, you had the opportunity to use your CSS and HTML skills to build a complete mock-up of a page design for a fictional Web site. You saw that combining CSS with standard HTML lets you build a Web design that follows good coding standards without sacrificing visually pleasing design. As you progressed through the building of the page, you saw how Web design is an interactive process that demands you continually

test your work to ensure compatibility across different browser platforms.

- The design process is iterative. Solicit feedback from all project stakeholders, and be prepared to create a number of design iterations before reaching consensus.
- You can use CSS properties to create interesting designs, such as the small feature boxes, without resorting to extra graphics.
- You can use CSS to control fonts, text alignment, color, and leading to gain complete control over your page typography.
- With CSS box model properties, you gain precise control over the white space within content columns and between different elements in a page design.
- The CSS page layout properties let you create navigation bars and column-based layouts.
- Use a wrapper for fixed page layouts to auto-center a page regardless of the user's resolution or monitor size.

Review Questions

1. What is the benefit of applying CSS rules to standard HTML markup?

2. What properties can you use to make standard heading elements, such as <h1> or <h2>, more distinctive?

3. In the small feature boxes section, what is the benefit of using descendant selection to style the elements that reside within the smfeature class division?

4. Why is it a good practice to state a background color for a Web page even when you are choosing white as the color?

5. What is the benefit of adding a line height setting to your standard content paragraphs, as you did in the feature article section?

6. When specifying a font family, what is the benefit of using a font substitution value?

7. What is the benefit of using the em as a measurement value, especially for fonts?

8. What is the benefit of working with visible element borders when you are designing a Web page?

9. What are two methods of making sure that floating page layouts are displayed correctly?

10. Why would you use a page wrapper division in a fixed page layout?

Hands-On Projects

1. Build the DogWorld page from this chapter using CSS to create a flexible layout. You can use the same graphics and content you used to build the fixed layout in this chapter. Refer to the "Building Flexible Layouts" section in Chapter 7 for ideas and guidance.

2. Build a different look for the DogWorld home page, using the graphics and content provided. How else can you arrange the layout to create an interesting and accessible design?

3. Build a secondary page for the DogWorld layout that is designed for reading rather than scanning. Use the graphical elements and content from the main page. Make sure to design for a smooth transition between pages, using some of the same look and feel that the main page exhibits.

Individual and Team Case Project

Finalize your project Web site by testing the finished design in multiple browsers and making any necessary adjustments or changes to support compatibility. If available, post your Web site to the class Web hosting area for live testing on the Internet. If you cannot post your Web site to the Web, prepare a CD or flash drive with your content to submit to your instructor.

If possible, enlist three to five people to review your Web site and fill out either your online or paper-based user feedback form. Compile the results and write a short paper detailing the results of

your testing and what they indicate about the effectiveness of your design. Point out the areas that you feel could benefit from user recommendations. List any assumptions you made about the Web site and how users either confirmed or denied these assumptions.

If the time and format allows, prepare to present and defend your Web design to your fellow students. Prepare a short oral presentation (10–15 minutes) where you can present your design ideas and reasons for building the site. Be prepared to take questions and defend your design ideas.

APPENDIX A

HTML5 Reference

This appendix provides the following information:

- Alphabetical HTML5 reference
- Obsolete elements
- Global attributes
- Event attributes
- Numeric and character entities

This appendix includes element descriptions sorted alphabetically. The elements listed in this appendix are the ones you will use most often, including a list of the global attributes allowed with the majority of HTML5 elements and a complete list of character entities. Some elements are obsolete in HTML5 but are still in common use in many Web sites. If you are creating HTML5-compliant Web pages, you should not use these obsolete elements. For more detailed information, visit the World Wide Web Consortium Web site at *www.w3.org*.

For more information on HTML5 elements, see: W3C Working Draft 9 November 2010 *www.w3.org/TR/html-markup*

The HTML5 draft specification was published as a First Public Working Draft on January 23, 2008 and several updated Working Drafts of the specification have subsequently been published by the W3C HTML5 working group. Information about HTML5 elements may continue to change as the draft evolves until it becomes a complete W3C recommendation.

Alphabetical HTML5 Reference

All elements support the global and event attributes described in Tables A-3 and A-4. Table A-2 contains a list of the obsolete elements.

Element	New in HTML5	Description	Attributes
<!–comment text–>		Insert a comment in your code Browsers do not display comments in the Web page Place the comment within the tag, for example: <!– This is a comment –>	None
<a>		Create a clickable hypertext anchor in a document; can be text or an image	Global attributes plus: *href* Target destination of the hyperlink *name* Name of a fragment of the document *target* Window or frame in which the linked document is displayed *hreflang* Language of the destination of the hyperlink *rel* Defines the relationship between the document and the destination *media* Valid media-query type *type* MIME type

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<abbr>		Abbreviation or acronym	Global attributes
<address>		Contact information	Global attributes
<area>		Clickable area in an image map	Global attributes plus: *alt* Specify an alternate string of text if the image cannot be displayed by the browser *href* Target destination of the hyperlink *hreflang* Language of the destination of the hyperlink *target* Window or frame in which the linked document is displayed *rel* Defines the relationship between the document and the destination *media* Valid media-query type *type* MIME type *shape* Determines the shape of the clickable area, circle or poly and coordinates
<article>	√	A section of content that forms an independent part of a document or Web site	Global attributes
<aside>	√	Content that is tangentially related to the main content	Global attributes
<audio>	√	Contains an audio stream	*src* URL of the radio stream content *autoplay* When set to *autoplay*, the video starts playing as soon as possible *preload* Indicates whether preloading is necessary *controls* When set to *controls*, displays playback controls *loop* When set to *loop*, the video plays repeatedly

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<b>		Boldface text This element is redefined in HTML5 to have semantic meaning: "Bold text is offset from its surrounding content without conveying any extra emphasis or importance, and for which the conventional typographic presentation is bold text"	Global attributes
<base>		Sets the base URL or target for a page This is an empty element	Global attributes plus: *href* Absolute or relative original URL for the current document *target* Default window or frame in which links contained in the document are displayed
<blockquote>		Indents text on both the left and right margins	
<body>		Identifies the body section of the Web page	
 		Inserts a line break, forcing text to the next line This is an empty element	
<button>		Creates a button outside of a form	Global attributes plus: *type* Values are button, reset, or submit *name* Name for the button *value* Initial value for the button; this can be changed by a script *disabled* When value is *disabled*, the button is not displayed *form* Value of the id attribute associated with the form
canvas	√	Used with scripting applications to dynamically render graphics, animations, or other visual images	Global attributes plus: *height* Height of the canvas, in pixels *width* Width of the canvas, in pixels
<caption>		Indicates that the text appears as the caption of a table	Global attributes

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<cite>		Title of a cited work	Global attributes
<code>		Fragment of computer code	Global attributes
<col>		One or more columns in a column group within the <colgroup> element This is an empty element	Global attributes plus: *span* Number of columns spanned
<colgroup>		Group of columns within a table	Global attributes plus: *span* Number of columns spanned
<command>	√	Defines a command button within a <menu> element This is an empty element	Global attributes plus: *type* Command, checkbox, or radio
<datalist>	√	Defines a list of option values for a form input element The results of this element are not displayed in the browser window	Global attributes
<dd>		Definition for a term within a definition list <dl> element	Global attributes
<del>		Text that has been deleted from a document	Global attributes
<details>	√	Represents a control from which a user can request more information	Global attributes plus: *open* Specifies the contents should be shown to the user, value is *open* or *null*
		Indicates a division within the document	Global attributes
<dt>		Term or name within a definition list <dl> element	
		Emphasizes text, usually as italic; browser determines the text style	Global attributes
<embed>	√	Contains content from an external source This is an empty element	*src* Address of the content being embedded *type* MIME type of the embedded content *height* Height of the embedded content, in pixels *width* Width of the embedded content, in pixels

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<fieldset>		Container for form controls grouped under a heading	Global attributes
<figcaption>	√	Caption or legend for a figure	Global attributes
<figure>	√	Self-contained set of graphical or related content	Global attributes
<footer>	√	Contains footer content	Global attributes
<form>		Contains user-submittable form	*action* URL of the application that processes the form data; this URL points to a script file or a e-mail address *enctype* Content type used to submit the form to the server (when the value of the method is "post"); most forms do not need this attribute *method* Specifies the HTTP method used to submit the form data; the default value is *get* • *get* Form data is appended to the URL specified in the action attribute • *post* Form data is sent to the server as a separate message *accept* Comma-separated list of content types that a server processing this form can handle correctly; most forms do not need this attribute *accept-charset* List of allowed character sets for input data that is accepted by the server processing this form; most forms do not need this attribute *name* Name of this form for data processing
<h1> to <h6>		Defines headings for six levels of content	Global attributes
<head>		Identifies the head section of the Web page, which is reserved for metadata (information about the document)	Global attributes
<header>	√	Header of a section	Global attributes

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<hgroup>	√	Group of headings <h1> through <h6>	Global attributes
<hr>		Inserts a horizontal rule on the page indicating thematic break in the content This is an empty element	Global attributes
<html>		Root element of an HTML file	Global attributes
<i>		Italicize text This element is redefined in HTML5 to have semantic meaning: "Italic text is offset from its surrounding content without conveying any extra emphasis or importance, and for which the conventional typographic presentation is italic text"	Global attributes
<iframe>		Creates an inline frame that contains a new content page	Global attributes plus: *src* Address of a page that the nested frame contains *name* Content name *width* Width of the iframe, in pixels *height* Height of the iframe, in pixels
<img>		Inserts an image into a Web page This is an empty element	Global attributes plus: *width* Width of the image, in pixels *height* Height of the image, in pixels *src* URL that points to the image file; this attribute is required *alt* Specify an alternate string of text if the image cannot be displayed by the browser
<input>		Used in a <form> element to create a variety of input controls	Global attributes plus: *text* Text entry field where the user entry is masked by asterisks *checkbox* Provides on/off toggles that the user selects

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
			radio Lets a user choose one value from a range of values *submit* Sends the form data to the server *reset* Clears the form of any user-entered data and returns it to its original state *hidden* Adds a control that is not displayed in the browser *image* Adds a graphic button to the form, rather than the default button *button* Creates a button that has no default behavior *file* Lets the user select a file that is submitted with the form *datetime* Global date and time *datetime-local* Local date and time *date* Calendar date *month* Calendar month *time* Time value *week* Calendar week *number* Number value *range* Range of values *email* Email address *url* URL value *search* Search term *tel* Telephone number *color* Color name
<ins>		Text that has been added to a document	Global attributes plus: *cite* URL of a text citation for the document *datetime* Date and time when the text was added
<kbd>		Represents user input	Global attributes

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<keygen>	√	Defines a key for secure communications	Global attributes plus: *challenge* Challenge text string that is submitted along with the key *keytype* Type of key, such as *rsa* *autofocus* Focuses on the textarea when the page loads *name* Name of this element for form processing *form* Value of the id attribute associated with the form *disabled* When value is *disabled*, the textarea is not displayed
<label>		Caption for a form control	Global attributes plus: *for* Reference the id of a form control to associate the label *form* Value of the id attribute associated with the form
<legend>		A title or explanatory caption for the parent element's content	Global attributes
		Marks an individual list item This is an empty tag	Global attributes
<link>		Defines a relationship between the document and external resources, such as a style sheet	Global attributes plus: *type* Type of external resource *href* URL of the external resource *rel* Describes the relationship between the current document and the anchor specified by the href attribute *hreflang* Language of the destination link *media* Valid media-query type
<map>		Defines an image map	Global attributes plus: *name* Name for the map
<mark>		Text marked or highlighted for reference purposes	Global attributes
<menu>		Redefined in HTML5 to define a menu	Global attributes plus: *type* Type of menu to display; values are context, toolbar, or list, list is the default

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<meta>		Used within the document head to provide information	*name* Meta information name, such as keyword or description *content* Content of the named information type
<meter>	√	Measurement or fractional value to measure data within a range	Global attributes plus: *value* Measured value shown by meter *min* Lower bound of the range for the meter *low* Point that marks the upper boundary of the *low* segment of the meter *high* Point that marks the lower boundary of the *high* segment of the meter *max* Upper bound of the range for the meter *optimum* Point that marks the *optimum* position for the meter
<nav>	√	Group of navigation links	Global attributes
<noscript>		Fallback content for scripts	Global attributes
<object>		Contains external content	Global attributes plus: *data* URL of the content *type* MIME type of the content *height* Height of the object, in pixels *width* Width of the object, in pixels *name* Name of the object
		Creates a numbered list	Global attributes
<optgroup>		Group of <option> elements with a common label	Global attributes plus: *label* Name of the group of options
<option>		Option choice in an <optgroup>	Global attributes plus: *label* Name of the group of options *value* Value for the option
<output>	√	Result of a calculation in a form	Global attributes plus: *name* Name of this element for form processing *form* Value of the id attribute associated with the form

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<p>		Marks the beginning of a new block of text	Global attributes
<param>		Defines parameters for plug-ins in <object> elements	Global attributes plus: *name* Name of the parameter *value* Value of the parameter
<pre>		Preserves the formatting and spacing of text as typed in the source code; displays the text in a monospace font, different from the standard browser text	Global attributes
<progress>	√	Indicates the completion progress of a task Used with a scripting language	Global attributes plus: *max* Total value at completion. *value* How much of the task has been completed
<q>		Quoted text from another source	Global attributes plus: *cite* Contains the address of the source of the quoted text
<rp>	√	Inserts ruby parentheses to hide ruby text <rt> content from browsers that do not support the <ruby> element	Global attributes
<rt>	√	Marks ruby text	Global attributes
<ruby>	√	Marks spans of content as ruby annotations Ruby text is used to provide a short annotation of the associated base text Ruby annotations are used frequently in Japan in many kinds of publications, including books and magazines Ruby is also used in China, especially in schoolbooks	Global attributes
<samp>		Sample computer code	Global attributes

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<script>		Contains dynamic script or data content	Global attributes plus: *type* MIME type of the script or data language *src* Address of the external script *defer* If set to *defer*, the script is executed after the page is loaded *charset* Character encoding of the external script
<section>	√	Section of a document typically including a title or heading	Global attributes
<select>		Form control for selecting from a list of options	Global attributes plus: *name* the name of this element for form processing *form* Value of the id attribute associated with the form *disabled* When value is *disabled*, the textarea is not displayed *size* Number of options to show *multiple* If set to *multiple*, user can select one or more options from the list
<small>		Contains legal, privacy, and other "fine-print" content	Global attributes
<source>	√	Allows specification for multiple media sources for audio and video elements	Global attributes plus: *src* Address of the media source *type* MIME type of the content *media* Valid media-query type
		Serves as an inline division; used to apply a style class or rule to text	Global attributes
		Contains important text, usually displayed as boldface; browser determines the text style	Global attributes
<style>		Used in the <head> section to contain CSS style rules	*type* Valid MIME type that designates a styling language *media* Valid media-query type specifies to which media the style applies

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<sub>		Subscripted text	Global attributes
<summary>		Summary, caption, or legend for a <details> element	Global attributes
<sup>		Superscripted text	Global attributes
<table>		Marks the beginning and end of a table	Global attributes plus: *summary* Description of the table
<tbody>		Block of rows in a table that contains the body content	Global attributes
<td>		Marks a data cell in a table	Global attributes plus: *colspan* Number of adjacent columns spanned by the <td> element *rowspan* Number of following rows spanned by the <td> element
<textarea>		Form control that lets users enter multiple lines of text content	Global attributes plus: *autofocus* Focuses on the textarea when the page loads *cols* Number of characters visible horizontally *disabled* When value is *disabled*, the textarea is not displayed *form* ID value defines the form the textarea belongs *maxlength* Defines the maximum number of characters *required* Element is a required part of the form submission *placeholder* Short text hint to aid the user when entering data *rows* Number of lines of text to display *wrap* When set to *hard*, the text wraps based on the *cols* value
<tfoot>		Block of rows in a table that contains the footer content	Global attributes
<th>		Forces the contents of a cell to be displayed as bold and centered	Global attributes

Table A-1 Common HTML5 Elements *(continues)*

(continued)

Element	New in HTML5	Description	Attributes
<thead>		Block of rows in a table that contains the header content	Global attributes
<time>	√	Contains a date or time value	Global attributes plus: *datetime* Specifies the date or time the element represents *pubdate* Indicates that the date or time given is a publication date or time
<title>		Specifies the title of the Web page; title text appears in the browser title bar and as the bookmark or favorites text	Global attributes
<tr>		Marks a row of cells in a table	Global attributes
<tt>		Specifies monospace text, usually Courier	Global attributes
		Creates a bulleted indented list	Global attributes
<var>		Variable in a mathematical expression	
<video>	√	Video or movie content	Global attributes plus: *autoplay* When set to *autoplay*, the video starts playing as soon as possible *preload* Indicates whether preloading is necessary. *controls* When set to *controls*, displays playback controls *loop* When set to *loop*, the video plays repeatedly *poster* URL for an image displayrf when the video is loading *height* Height of the video, in pixels *width* Width of the video, in pixels
<wbr>	√	Represents an approved line-break location	Global attributes

Table A-1 Common HTML5 Elements

Obsolete Elements

Table A-2 lists the elements that are no longer supported in HTML5.

Element	HTML5 Replacement
<acronym>	<abbr>
<applet>	<object>
<basefont>	Use CSS
<big>	Use CSS
	Use CSS
<dir>	
<font>	Use CSS
<frame>	No replacement
<frameset>	No replacement
<noframes>	No replacement
<s>	Use CSS
<strike>	Use CSS
<tt>	Use CSS
<u>	Use CSS
<xmp>	<pre>

Table A-2 Obsolete Elements

Global Attributes

Table A-3 lists the global attributes, which are allowed within all of the elements listed in the element tables.

Attribute	New in HTML5	Definition
accesskey		Key label or list of key labels with which to associate the element; each key label represents a keyboard shortcut which browsers can use to activate the element or give focus to the element
class		Specifies a class name for an element; the class name can be used to specify style sheet rules
contenteditable	√	Specifies whether the contents of the element are editable Values are *true*, *false*, or empty

Table A-3 HTML5 Global Attributes *(continues)*

(continued)

Attribute	New in HTML5	Definition
contextmenu	√	Use with the <menu> element to indicate that the element is part of a context menu
dir		Specifies the element's text directionality Values are *ltr*, (left to right) or *rtl* (right to left)
draggable	√	Specifies whether the element is draggable Values are *true* or *false*
hidden	√	Specifies that the element represents an element that is not yet, or is no longer, relevant
id		Specifies a document-wide unique identifier for an element Multiple elements in the document cannot have the same id value
lang		Specifies the primary language for the contents of the element and for any of the element's attributes that contain text
spellcheck	√	Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking Values are *true*, *false*, or empty
style		Specifies a style sheet rule for the element
tabindex		Specifies whether the element represents an element that is focusable (that is, an element which is part of the sequence of focusable elements in the document), and the relative order of the element in the sequence of focusable elements in the document
title		Specifies a title for the element; the content of the title is displayed in the browser as a pop-up window or tooltip

Table A-3 HTML5 Global Attributes

Event Attributes

Table A-4 lists the event attributes, which are used with a scripting language to trigger events.

Attribute	Script is Triggered
onafterprintNew	After the document is printed
onbeforeonloadNew	Before the document loads
onbeforeprintNew	Before the document is printed
onblur	When the window loses focus
onerrorNew	When an error occurs
onfocus	When the window gets focus
onhaschangeNew	When the document has change
onload	When the document loads
onmessageNew	When the message is triggered
onofflineNew	When the document goes offline
ononlineNew	When the document comes online
onpagehideNew	When the window is hidden
onpageshowNew	When the window becomes visible
onpopstateNew	When the window's history changes
onredoNew	When the document performs a redo
onresizeNew	When the window is resized
onstorageNew	When a document loads
onundoNew	When a document performs an undo
onunloadNew	When the user leaves the document

Table A-4 HTML5 Event Attributes

Numeric and Character Entities

Table A-5 lists the event attributes, which are used with a scripting language to trigger events.

Character	Character Entity	Numeric Entity	Description
"	{	"	Quotation mark
#	#		Number sign
$	$		Dollar sign
%	%		Percent sign
&	&	&	Ampersand

Table A-5 Numeric and Character Entities *(continues)*

(continued)

Character	Character Entity	Numeric Entity	Description
‘	'		Apostrophe
(	(		Left parenthesis
)	)		Right parenthesis
*	*		Asterisk
+	+		Plus sign
,	,		Comma
-	-		Hyphen
.	.		Period (full stop)
/	/		Solidus (slash)
0	0		Digit 0
1	1		Digit 1
2	2		Digit 2
3	3		Digit 3
4	4		Digit 4
5	5		Digit 5
6	6		Digit 6
7	7		Digit 7
8	8		Digit 8
9	9		Digit 9
:	:		Colon
;	;		Semicolon
<	<	<	Less than sign
	=		Equals sign
>	>	>	Greater than sign
?	?		Question mark
@	@		Commercial at
A Z	A - Z		Uppercase letters A Z
[	[		Left square bracket
\	\		Reverse solidus (backslash)
]	]		Right square bracket
ˆ	^		Caret
_	_		Horizontal bar (underscore)
`	`		Grave accent
a z	a - &#122		Lowercase letters a z

Table A-5 Numeric and Character Entities *(continues)*

(continued)

Character	Character Entity	Numeric Entity	Description
{	{		Left curly brace
|				Vertical bar
}	}		Right curly brace
~	~		Tilde
			Nonbreaking space
¡	¡	¡	Inverted exclamation mark
¢	¢	¢	Cent sign
£	£	£	British Pound sign
$	¤	¤	Currency sign
¥	¥	¥	Yen sign
‖	¦	¦	Broken vertical bar
§	§	§	Section sign
¨	¨	¨	Spacing diaeresis
©	©	©	Copyright sign
a	ª	ª	Feminine ordinal indicator
«	«	«	Left-pointing double angle quotation mark
¬	¬	¬	Not sign
_			Soft hyphen
®	®	®	Registered trademark sign
¯	¯	¯	Macron overline
°	°	°	Degree sign
±	±	±	Plus-or-minus sign
2	²	²	Superscript digit 2
3	³	³	Superscript digit 3
´	´	´	Acute accent
µ	µ	µ	Micron sign
¶	¶	¶	Paragraph sign
·	·	·	Middle dot
¸	¸	¸	Cedilla
1	¹	¹	Superscript digit 1
o	º	º	Masculine ordinal indicator
»	»	»	Right-pointing double angle quotation mark
¼	¼	¼	Fraction one-quarter
½	½	½	Fraction one-half

Table A-5 Numeric and Character Entities *(continues)*

(continued)

Character	Character Entity	Numeric Entity	Description
¾	¾	¾	Fraction three-quarters
¿	¿	¿	Inverted question mark
À	À	À	Capital letter A with grave
Á	Á	Á	Capital letter A with acute
Â	Â	Â	Capital letter A with circumflex
Ã	Ã	Ã	Capital letter A with tilde
Ä	Ä	Ä	Capital letter A with diaeresis
Å	Å	Å	Capital letter A with ring above
Æ	Æ	&Aelig;	Capital letter AE
Ç	Ç	Ç	Capital letter C with cedilla
È	È	È	Capital letter E with grave
É	É	É	Capital letter E with acute
Ê	Ê	Ê	Capital letter E with circumflex
Ë	Ë	Ë	Capital letter E with diaeresis
Ì	Ì	Ì	Capital letter I with grave
Í	Í	Í	Capital letter I with acute
Î	Î	Î	Capital letter I with circumflex
Ï	Ï	Ï	Capital letter I with diaeresis
D	Ð	Ð	Capital letter ETH
Ñ	Ñ	Ñ	Capital letter N with tilde
Ò	Ò	Ò	Capital letter O with grave
Ó	Ó	Ó	Capital letter O with acute
Ô	Ô	Ô	Capital letter O with circumflex
Õ	Õ	Õ	Capital letter O with tilde
Ö	Ö	Ö	Capital letter O with diaeresis
x	×	×	Multiplication sign
Ø	Ø	Ø	Capital letter O with stroke
Ù	Ù	Ù	Capital letter U with grave
Ú	Ú	Ú	Capital letter U with acute
Û	Û	Û	Capital letter U with circumflex
Ü	Ü	Ü	Capital letter U with diaeresis
Ý	Ý	Ý	Capital letter Y with acute
P	Þ	Þ	Capital letter THORN
ß	ß	ß	Sz ligature

Table A-5 Numeric and Character Entities *(continues)*

(continued)

Character	Character Entity	Numeric Entity	Description
à	à	à	Small letter a with grave
á	á	á	Small letter a with acute
â	â	â	Small letter a with circumflex
ã	ã	ã	Small letter a with tilde
ä	ä	ä	Small letter a with diaeresis
å	å	å	Small letter a with ring above
æ	æ	æ	Small letter ae
ç	ç	ç	Small letter c with cedilla
è	è	è	Small letter e with grave
é	é	é	Small letter e with acute
ê	ê	ê	Small letter e with circumflex
ë	ë	ë	Small letter e with diaeresis
ì	ì	ì	Small letter i with grave
í	í	í	Small letter i with acute
î	î	î	Small letter i with circumflex
ï	ï	ï	Small letter i with diaeresis
d	ð	ð	Small letter eth
ñ	ñ	ñ	Small letter n with tilde
ò	ò	ò	Small letter o with grave
ó	ó	ó	Small letter o with acute
ô	ô	ô	Small letter o with circumflex
õ	õ	õ	Small letter o with tilde
ö	ö	ö	Small letter o with diaeresis
÷	÷	÷	Division sign
o/	ø	ø	Small letter o with stroke
ù	ù	ù	Small letter u with grave
ú	ú	ú	Small letter u with acute
û	û	û	Small letter u with circumflex
ü	ü	ü	Small letter u with diaeresis
ý	ý	ý	Small letter y with acute
þ	þ	þ	Small letter thorn
ÿ	ÿ	ÿ	Small letter y with diaeresis

Table A-5 Numeric and Character Entities

APPENDIX B

CSS Reference

This appendix provides the following information:

- CSS notation reference
- Alphabetical CSS property reference
- CSS measurement units

This appendix includes all CSS2 properties and the more commonly supported CSS3 property descriptions, sorted both alphabetically and by category. For more detailed information, visit the World Wide Web Consortium Web site at *www.w3.org*.

CSS Notation Reference

Table B-1 lists the notation symbols used in CSS properties.

Notation	Definition
<>	Words between angle brackets specify a type of value; for example, <color> means to enter a color value such as red
|	A single vertical bar between values means one or the other must occur; for example, scroll | fixed means choose scroll or fixed
||	Two vertical bars separating values means one or the other or both values can occur; for example, <border-width> || <border-style> || <color> means any or all of the three values can occur
[ ]	Square brackets group parts of the property value together; for example, none | [underline || overline || line-through || blink] means the value is either none or one of the values within the square brackets

Table B-1 CSS Notations

Alphabetical CSS Property Reference

Table B-2 describes the CSS properties in alphabetical order.

Property	New in CSS 3	Values	Default	Applies to	Inherited
Background		<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>	None	All elements	No
Background-attachment		scroll | fixed	scroll	All elements	No
Background-color		color name or hexadecimal value | transparent	transparent	All elements	No
Background-image		<url> | none	none	All elements	No
Background-position		[<percentage> | <length>] {1,2} | [top | center | bottom] || [left | center | right]	0% 0%	Block-level and replaced elements	No

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Background-repeat		repeat | repeat-x | repeat-y | no-repeat	repeat	All elements	No
Border		<border-width> || <border-style> || <color>	None	All elements	No
Border-bottom		<border-bottom-width> || <border-style> || <color>	None	All elements	No
Border-bottom-color		<color>	Value of the color property	All elements	No
Border-bottom-left-radius	√	[<length> | <percentage>] [<length> | <percentage>]	0	All elements	No
Border-bottom-right-radius	√	[<length> | <percentage>] [<length> | <percentage>]	0	All elements	No
Border-bottom-style		none | dotted | dashed | solid | double | groove | ridge | inset | outset	none	All elements	No
Border-bottom-width		thin | medium | thick | <length>	medium	All elements	No
Border-collapse		separate | collapse	separate	<table> element	Yes
Border-color		<color>	Value of the color property	All elements	No
Border-left		<border-left-width> || <border-style> || <color>	None	All elements	No
Border-left-color		<color>	Value of the color property	All elements	No
Border-left-style		none | dotted | dashed | solid | double | groove | ridge | inset | outset	none	All elements	No
Border-left-width		thin | medium | thick | <length>	medium	All elements	No
Border-radius	√	[<length> | <percentage>] {1,4} [/ [<length> |<percentage>]{1,4}]	0	All elements	No
Border-right		<border-right-width> || <border-style> || <color>	None	All elements	No

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Border-right-color		<color>	Value of the color property	All elements	No
Border-right-style		none | dotted | dashed | solid | double | groove | ridge | inset | outset	none	All elements	No
Border-right-width		thin | medium | thick | <length>	medium	All elements	No
Border-style		none | dotted | dashed | solid | double | groove | ridge | inset | outset	none	All elements	No
Border-top		<border-top-width> || <border-style> || <color>	None	All elements	No
Border-top-color		<color>	Value of the color property	All elements	No
Border-top-left-radius	√	[<length> | <percentage>] [<length> | <percentage>]	0	All elements	No
Border-top-right-radius	√	[<length> | <percentage>] [<length> | <percentage>]	0	All elements	No
Border-top-style		none | dotted | dashed | solid | double | groove | ridge | inset | outset	none	All elements	No
Border-top-width		thin | medium | thick | <length>	medium	All elements	No
Border-width		[thin | medium | thick | <length>]	None	All elements	No
Bottom		<length> | <percentage> | auto	auto	Positioned elements	
Box-shadow	√	none | <shadow> [, <shadow>]*	None	All elements	No
Caption-side		top | bottom	top	<caption> element	Yes
Clear		none | left | right | both	none	All elements	No
Color		<color>	Browser specific	All elements	Yes

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Display		inline | block | list-item | run-in | compact | marker | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none	inline	All elements	No
Float		left | right | none	none	All elements	No
Font		[<font-style> || <font-variant> || <font-weight>] <font-size> [/ <line-height>] <font-family>	None	All elements	Yes
Font-family		Font family name (such as Times) or generic family name (such as sans-serif)	Browser specific	All elements	Yes
Font-size		<absolute-size> | <relative-size> | <length> | <percentage>	medium	All elements	Yes
Font-size-adjust	√	<number> | none | inherit	None	All elements	Yes
Font-stretch		normal | wider | narrower | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded	normal	All elements	Yes
Font-style		normal | italic | oblique	normal	All elements	Yes
Font-variant		normal | small caps	normal	All elements	Yes
Font-weight		normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900	normal	All elements	Yes
Height		<length> | <percentage> | auto	auto	Block-level and replaced elements; also all elements except inline images	No
Left		<length> | <percentage> | auto	auto	Positioned	No

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Letter-spacing		normal | <length>	normal	All elements	Yes
Line-height		normal | <number> | <length> | <percentage>	normal	All elements	Yes
List-style		<keyword> || <position> || <url>	None	Elements with display value list-item	Yes
List-style-image		<url> | none	none	Elements with display value list-item	Yes
List-style-position		inside | outside	outside	Elements with display value list-item	Yes
List-style-type		disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none	disc	Elements with display value list-item	Yes
Margin		[<length> | <percentage> | auto]	None	All elements	No
Margin-bottom		<length> | <percentage> | auto	0	All elements	No
Margin-left		<length> | <percentage> | auto	0	All elements	No
Margin-right		<length> | <percentage> | auto	0	All elements	No
Margin-top		<length> | <percentage> | auto	0	All elements	No
Max-Width	√	<length> | <percentage> | auto	auto	All elements but inline text elements and table rows	No
Min-Width	√	<length> | <percentage> | auto	auto	All elements but inline text elements and table rows	No
Opacity	√	<alphavalue> | inherit	1	All elements	No
Overflow	√	[visible | hidden | scroll | auto]	visible	All elements	No
Padding		<length> | <percentage>	0	All elements	No
Padding-bottom		<length> | <percentage>	0	All elements	No
Padding-left		<length> | <percentage>	0	All elements	No
Padding-right		<length> | <percentage>	0	All elements	No
Padding-top		<length> | <percentage>	0	All elements	No

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Position		static | relative | absolute | fixed	static	All elements except generated content	No
Right		<length> | <percentage> | auto	auto	Positioned elements	No
Text-align		left | right | center | justify	Depends on browser and language direction	Block-level elements	Yes
Text-decoration		none | [underline || overline || line-through || blink]	none	All elements	No
Text-decoration-color	√	<color>	Current color	All elements	No
Text-decoration-style	√	solid | double | dotted | dashed | wave	Solid	All elements	No
Text-indent		<length> | <percentage>	0	Block-level	Yes
Text-outline	√	none | [<color> <length> <length>? | <length> <length>? <color>]	none	All elements	Yes
Text-shadow		none | [<color> || <length> <length> <length>? ,]* [<color> ||<length> <length> <length>?]	none	All elements	Yes
Text-transform		capitalize | uppercase | lowercase | none	none	All elements	No
Text-wrap	√	normal | unrestricted | none | suppress	normal	All elements	Yes
Top		<length> | <percentage> | auto	auto	Positioned	
Vertical-align		baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>	baseline	Inline elements	No
White-space		normal | pre | nowrap	normal	Block-level elements	

Table B-2 CSS Properties *(continues)*

(continued)

Property	New in CSS 3	Values	Default	Applies to	Inherited
Width		<length> | <percentage> | auto	auto	All elements but inline text elements and table rows	No
Word-spacing		normal | <length>	normal	All elements	Yes
Word-wrap	√	normal | break-word	normal	All elements	Yes
Z-index		auto | integer	auto	Positioned elements	No

Table B-2 CSS Properties

CSS Measurement Units

Table B-3 defines the CSS measurement units.

Unit	Code Abbreviation	Description
Centimeter	cm	Standard metric centimeter
Em	em	Width of the capital M in the current font, usually the same as the font size
Ex	ex	Height of the letter x in the current font
Inch	in	Standard U.S. inch
Millimeter	mm	Standard metric millimeter
Pica	pc	Standard publishing unit equal to 12 points
Pixel	px	Size of a pixel on the current display
Point	pt	Standard publishing unit; 72 points in an inch
Relative	For example: 150%	Sets a font size relative to the base font size; 150% equals 1.5 times the base font size

Table B-3 CSS Measurement Codes

APPENDIX C

CSS Media Queries

This appendix provides the following information:

- Overview of CSS Media Queries
- Supported media types
- Media features

CSS3 introduces Media Queries, an expansion on the concept of media types in CSS2. With media types you could specify different style rules for different types of destination media, such as screen or print, without changing the content for each device. Media types are commonly supported by all browsers and let you specify different styles for common destination media, such as *print* or *screen*. Media Queries let you create much more precise rules for different types of destination media.

Overview of CSS Media Queries

A Media Query contains both a media type and optional expressions that check conditions called media features. The media features include characteristics such as the width or height of the destination device. For example you could specify a certain font when a device screen width is 480 pixels, the screen width of the iPhone, or create a separate style sheet for color or monochrome printers.

Media Queries are not evenly supported by the major browsers. Test carefully to make sure the queries you write work across all browsers.

Media Query Syntax

Media Queries combine a media type and a media feature. In the following example the media type is *screen* and the media feature is *max-device-width*. The media feature is set to a value of 480 pixels. The syntax for the Media Query is like the standard CSS property syntax.

```
media="screen and (max-device-width: 480px)"
```

You can add multiple media features with the *and* keyword, as shown in the following code:

```
media="screen and (min-width:800px) and (max-width:1280px)"
```

Applying Media Queries

There are three ways to apply Media Queries to specify style rules. In this first example, the link element specifies an external style sheet named mobile-device.css that will be applied if the device is a screen with a maximum width of 480 pixels.

```
<link rel="stylesheet" type="text/css" media="screen and
   (max-device-width: 480px)"
  href="mobile_device.css" />
```

The next example uses an at-import rule to specify an external style sheet named mobile-device.css that will be applied if the device is a screen with a maximum width of 480 pixels.

```
@import url("mobile-device.css") screen and (max-device-width:
   480px);
```

This final example uses the @media rule within a <style> element. Notice the external curly brackets that contain all of the style rules.

```
<style type="text/css">
@media screen and (max-device-width: 480px) {
  ...style rules..
  }
</style>
```

Supported Media Types

CSS3 supports the media types listed in Table C-1.

Media Type	Description
all	Suitable for all devices
braille	Intended for braille tactile feedback devices
embossed	Intended for paged braille printers
handheld	Intended for handheld devices (typically small screen, limited bandwidth)
print	Intended for paged material and for documents viewed on screen in print preview mode
projection	Intended for projected presentations, for example projectors
screen	Intended primarily for color computer screens
speech	Intended for speech synthesizers
	Note: CSS2 had a similar media type called *aural* for this purpose
tty	Intended for media using a fixed-pitch character grid (such as teletypes, terminals, or portable devices with limited display capabilities)
	Authors should not use pixel units with the *tty* media type
tv	Intended for television-type devices (low resolution, color, limited-scrollability screens, sound available)

Table C-1 CSS3 Media Types

Media Features

CSS3 supports the media features listed in Table C-2. Notice that many of the features have min- and max- prefixes to express constraints. These descriptions are courtesy of the W3C (*www.w3.org/TR/css3-mediaqueries/#media1*).

Feature	Description	Value
Width Min-Width Max-Width	Width of the targeted display area of the output device For print this is the printable width of the page; for devices this is the width of the viewport	Length
Height Min- Height-width Max- Height-width	Height of the targeted display area of the output device For print this is the printable height of the page; for devices this is the height of the viewport	Length
Device-width Min- Device-width Max- Device-width	Width of the screen of the output device For print this is the width of the page; for devices this is the width of the viewport	Length
Device-height Min- Device-height Max- Device-height	Height of the screen of the output device For print this is the height of the page; for devices this is the height of the viewport	Length
Orientation	Value is *portrait* when the length of the *height* media feature is greater than or equal to the length of the *width* media feature; otherwise the value is *landscape*.	Portrait | Landscape
Aspect-ratio Min- Aspect-ratio Max- Aspect-ratio	Ratio of the value of the *width* media feature to the value of the *height* media feature	Ratio For example, 16/9
Device-aspect-ratio Min- Device-aspect-ratio Max- Device-aspect-ratio	Ratio of the value of the 'device-width' media feature to the value of the 'device-height' media feature	Ratio
Color Min- Color Max- Color	Number of bits per color component of the output device If the device is not a color device, the value is zero	Integer
Color-index Min- Color-index Max- Color-index	Number of entries in the color lookup table of the output device If the device does not use a color lookup table, the value is zero	Integer
Monochrome Min- Monochrome Max- Monochrome	Number of bits per pixel in a monochrome frame buffer If the device is not a monochrome device, the output device value will be 0	Integer

Table C-2 Supported Media Features *(continues)*

(continued)

Feature	Description	Value
Resolution Min- Resolution Max- Resolution	Screen resolution of the output device	Resolution For example, 1024 x 768
Scan	Describes the scanning process of "tv" output devices	Progressive | Interlace
Grid	Query whether the output device is grid or bitmap If the output device is grid-based (e.g., a tty terminal, or a phone display with only one fixed font), the value will be 1; otherwise, the value will be 0	Integer

Table C-2 Supported Media Features

Examples of CSS Media Features

The font for screen output is sans-serif.

```
@media screen {
  body { font-family: sans-serif }
}
```

The font for print output is serif and the color is black.

```
@media print {
  body { font-family: serif; color; #000; }
}
```

The following link elements select different external style sheets based on the media type .

```
<link rel="stylesheet" type="text/css" media="screen"
   href="screen.css">
<link rel="stylesheet" type="text/css" media="print"
   href="print.css">
```

The style will apply if the screen is 800 pixels wide.

```
@media screen and (device-width: 800px) { ... }
```

The style will apply if the device has a landscape orientation.

```
@media screen and (orientation: landscape) { ... }
```

APPENDIX D

Print Style Sheets

This appendix provides the following information:

- Applying print styles
- Creating print styles

The print media type (described in Appendix C) lets you add style sheet rules specifically for the printed page. Users may want to print your page for a variety of reasons, and complex Web designs often do not print well. Link colors, background images, font sizes and other Web features do not translate well to the printed page. You can control how your printed pages look with a print style sheet.

Applying Print Styles

A print style sheet can be an external style sheet file, or the rules can be embedded in the head section of a document. In the following code, the <link> element specifies the media type print and points to a style sheet named print–styles.css. The rules in this style sheet will apply to the printed version of the Web page.

```
<link rel="stylesheet" type="text/css" media="print"
href="print_styles.css" />
```

Print style rules can also be embedded in the style section of the document using the @media keyword. Notice the additional external curly brackets that contain all of the style rules.

```
<style type="text/css">
@media print {
  body {
  font-size; 12pt;
  color: #000;}
  }
</style>
```

The external style sheet method of applying print rules is the simplest way to manage print styles. If you have an external style sheet for both screen and print, only one will apply based on the destination media. If you have internal styles and use the <link> element only for a print style sheet, you may need to use the !important keyword (described in Chapter 4) to override style rules. For example, if your internal style sheet contains rules that specify a serif font-family, and you want to override those styles for printing, you would use the !important keyword in the external print style sheet as shown below:

```
body {font-family: serif !important;}
```

Creating Print Styles

With print style rules you can change or remove any element on your Web page. You can control fonts, colors, borders, and backgrounds. You can also hide elements that are not relevant on the printed page, such as navigation links.

Specifying Fonts and Color

Standard font sizes for printed text are different than what you would specify onscreen and text prints best in black. Two simple style rules accomplish this:

```
body {font-size; 12pt;
      color: #000
  }
```

Specifying Background Colors

You may also want to set background-colors for the page to white for better legibility when printing. This rule would also apply to the body selector.

```
body {font-size; 12pt;
      color: #000
      background-color: white;
}
```

Removing Elements

You can turn off navigation or other elements, such as search forms, that are not relevant on a printed page. For example, in the dogworld.com project page (see Figure 12-6) the Search tool and Sponsored Links section in the right column are not necessary in print. Figure D-1 shows the print preview of the Web page in Internet Explorer. Notice that by default browsers omit background images in the printed version. Figure D-2 shows the print results after print style rules have been applied.

Figure D-1 Print Preview version of Dogworld page

To turn off the navigation links, set display to *none* for the division element that contains the links. In the Dogworld code, the navigation links are contained in the <ul> element with an id of *navlist*. The following style rule hides the navigation list.

```
#navlist {
   display: none;
}
```

You can also hide the entire right column, containing the search and links content, with one style rule that hides the entire right column containing both items.

```
div#rightcol {
   display: none;
}
```

Figure D-2 Print style sheet removes unwanted content from printed version of the Web page

Index

A

absolute font size keywords, 206
absolute measurement values, 251
absolute paths and partial URLS, 126
absolute units, 198
accept attribute, 484
accept-charset attribute, 484
accessibility
  access keys, 93
  constraints and audience, 119–120
  designing for, 92–97
  operable content, 96
  optional navigation links, 93
  perceivable content, 95–96
  robust content, 97
  table elements, 448
  understandable content, 96–97
  user-controlled font size, 93
  WCAG 2.0 guidelines, 94–97
access keys, 93
action attribute, 484
active link colors, 370
active pseudo-class, 370
active white space, **75**–76, 99
Adobe Creative Suite, 120
Adobe Dreamweaver, 29, 138
Adobe Fireworks, 346
Adobe HomeSite, 29
Adobe Illustrator, 346
Adobe Photoshop, 341, 346
Adobe Shockwave, 340
Adobe Web site, 120
Advanced GIF Animator, 340
<a> elements, 174, 408, 409, 411, 414, 417
  borders, 426
  class = "chapter" attribute, 432, 433
  display property, 429
  display type, 428
  href attributes, 406
   elements, 429, 430
  link text, 425
  name attribute, 411
  removing underlining, 220
   element, 436
  visited state, 433
:after pseudo-elements, 178–179
AIGA Web site, 433
AJAX (Asynchronous JavaScript and XML), **483,** 520
align attribute, 347–348
aligning
  form elements, 501–503
  text, 213–214
alphabetical system, 231
alt attribute, 95, 172, 346, 348, 420–421, 492
alternate
  color rows, 466
  fonts, 205
  text, 350
  text-based links, 420–421
Amaya, 121
Amazon.com Web site, 55, 57, 59
analogous color schemes, **363,** 384
animated GIFs, 339–340
animation
  GIF (Graphics Interchange Format), 339–340
  number of times played, 340
  time between frames, 340
Apache, 483
application developers, 122
applications and background processing, 28
area class, 516, 518
article division
  fixed layouts, 325–326
  formatting, 314
<article> element, 23, 26–27, 308
  fixed layouts, 317
  fixed width, 318
  nonfloating, 301
article style rule, 325
artwork, saving original copy, 342
Asian languages accent marks, 223
<aside> elements, 23, 27
aspect ratio, **351,** 384
.asp extension, 124
Asynchronous JavaScript and XML. *See* AJAX (Asynchronous JavaScript and XML)
attributes, 25
  matching parts of value, 182
  matching with values, 172
  presentation information, 25
attribute selectors, 172
audience, 51
  accessibility constraints, 119–120
  analyzing, 115–121
  captive, 117
  education, 116
  employees of organization, 117
  gathering information about, 115–121
  reasons for visiting Web site, 117
  software tools, 120–121
  technology issues, 116, 119–120
  typical members, 116
  Web analytics, **117**–119
audience definition, **115,** 144
audio, 28
<audio> elements, 23, 28

B

background color layer, 367
background-color property, 249, 367, 370–371, 434, 456
background colors, 370–371
  AIGA Web site, 433
  <body> element, 372
  changing on hover, 435–438
  colgroup class, 471–472
  column group, 456
  columns, 471
  content box, 283
   element, 372
  elements, 372
  forms, 505
  <h1> element, 373
  highlighting links, 438
  horizontal navigation bars, 426
  hover effect, 473
  indicating history, 432–433
  indicating location, 433–434
  navigation, 432–434
  padding area, 248, 255
  pages, 372–373
  paragraphs, 249
  specifying, 465–466
  styling table, 465–468
  <td> elements, 466

<th> elements, 466, 471
vertical navigation bar, 431
background graphics
horizontal repeating, 378–379
vertical repeating, 377–378
background hover effects, 467–468
background-image property, 367, 374, 433, 456, 507
background images
behavior, 377, 381
<body> element, 375
changing on hover, 436–437
controlling with CSS, 373–383
<fieldset> element, 506–507
nonrepeating, 379–380
positioning repeating, 382–383
selecting, 374–375
specifying position, 380–381
tiling, 374
URL, 375
background-position property, 380–381, 433, 507
background processing, 28
background properties, 456
background repeat, 377
background-repeat property, 377, 507
no-repeat value, 379–380
repeat-x value, 378–379
repeat-y value, 377–378
backgrounds
complementary colors, 362
pages, 375–376
tiling images, 375–377
Web pages, 375–376
Back to Top link, 409
backups of Web sites, 138
backward compatibility, 33
balloons_sm.jpg image, 350
banding, **359,** 384
banner division, 297
Barnes and Noble Web site, 348, 420–421
basic.htm file
copying, 157
opening, 158
basic style sheet, building, 157–159
basic tables, 449–450
BBEdit, 121
Bebo, 113
:before pseudo-elements class, 178–179
Berners-Lee, Tim, 9, 92
best class selector, 464
billboard Web sites, 112
block-level boxes, 245
block-level elements, 170, 247, 250
border-radius property, 267
as box with content, 248
consistent spacing, 254
:first-letter pseudo-element class, 177
lists, 424
margin, border, and padding values, 246
normal flow, 294
Blogarama Web site, 113
Blog Catalog Web site, 113
bloggers, 113
blogs, 113
bluebtn.jpg file, 437
Bluefish, 121
<body> element, 4
<body> elements, 22, 160–161, 246
background color, 372
background image, 375
color, 369
normal flow, 295
border attribute, 347–348
border-bottom-color property, 265
border-bottom-left-radius property, 268
border-bottom property, 258, 438
border-bottom-right-radius property, 268
border-bottom-style property, 261
border-bottom-width property, 263
border-collapse property, 459
border-color property, 264–266
shorthand notation, 265–266
border-left-color property, 265
border-left property, 258
border-left-style property, 261
border-left-width property, 263
border properties, **258**–268, 287, 353
border property, 249, 258
measurement values, 251
border radius properties, 268
border-radius property, 267–268
border-right-color property, 265
border-right property, 258
border-right-style property, 261
border-right-width property, 263
borders, 282
absolute value, 262
<a> element, 426
appearance, 258–268
block elements, 250
cells, 449, 451, 470
characteristics, 165
collapsing table borders, 451
colors, 264–266, 368
content box, 248–251
default, 259
elements, 262
horizontal navigation bars, 426
images, 348
individual styles, 261
relative value, 262
removing extra space between, 459
rounded, 267–268
shorthand properties, 266–267
style rules, 459–460
styles, 259–261
styling, 458–461
styling row and cell, 460
tables, 470
width, 262–264
border shortcut property, 165, 248
border-style property, 259–261, 266
border-style to double rule, 267
border-style to solid rule, 266, 267
border-top-color property, 265
border-top-left-radius property, 268
border-top property, 258
border-top-right-radius property, 268
border-top-style property, 261
border-top-width property, 263
border-width property, 262–264, 266
Bos, Bert, 207
Boston Vegetarian Society Web site, 63
boxes, 245
box model, **248**–251, 287
calculating width, 270
margin properties, **251**–255
padding area, 255–257
paragraph element, 248
box model.html file, 280
box model1.html file, 280
box properties, 296–297
applying, 280–286
tables, 461
box-shadow paragraph style rules, 279
box shadows, 278–280
box types, 245
breadcrumb path, **402,** 439
Broads Authority Web site, 60–62
BrowserNews Web site, 51
browsers
application support in, 14
centering pages in, 297–298
clearing cache, 53
compatibility issues, 50–51
CSS3, 152
CSS properties, 152
CSS support, 151–152
default fonts, 195, 203–204
displaying element content boxes, 245
downloading, 51
element display, 247
font-face property, 195, 205
fonts, 193
font-stretch property, 210
font weights, 209
HTML (Hypertext Markup Language), 5
justified text, 214
most popular, 51
older, 50–51
Open Type format, 195
operating systems, 54
platforms, 51
proprietary elements, 10
removing default margin spacing, 254–255
rendering engine, **5,** 50
small capitals, 208
standards-compliant, 51
style elements, 6
support for CSS, 7
support for new page layout elements, 28
SVG graphics, 343
testing, 5, 50–51, 140
TrueType format, 195
versions, 54
viewing source code, 4

browser-safe colors, **359,** 384
bulleted lists, 422
  customizing, 230–236
  default bullets, 423
  default padding and margins, 423
  images, 233
  list-item properties, 235–236
  list markers, 230–231, 234–235
bullets
  hiding, 247
  removing default, 423
button attribute, 486
buttons
  changing color, 436–437
  width and horizontal navigation bars, 428–429

C

cable modem, 136
cache, **53,** 99
  clearing, 53
<canvas> elements, 23, 28
capitalization, 220–221
  key words, 228–229
caps class, 228
<caption> elements, 448, **450,** 471, 475
captions
  font-family, 472–473
  input objects, 486–488
  radio buttons, 491
  styling, 457–458
  tables, 471
caption-side property, 457
Cartoon Network Web site, 78
cascade, **180**–181, 184
Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
case sensitivity and filenames, 123
catalog structure, 133–134
catalog Web sites, 114
cells, 449–450
  borders, 449, 451, 460, 470
  centering text, 452
  padding, 462, 470
  spanning, 451–452
  table data, 459
  table header, 459
CERN (European Laboratory for Particle Physics), 9
CGI (Common Gateway Interface), **483,** 520
chapter files, 416
Chapter04 folder, 157–158, 163
Chapter05 folder, 224
Chapter09 folder, 403–404, 413
Chapter 10 folder, 469
chapter1.html file, 403, 413, 416–418
chapter2.html file, 404, 408, 413
chapter3.html file, 404, 408, 413
Chapter 2 link, 412
chapter pages, linking, 407–408
chapters and topic links, 414
character exceptions, 124
character set, 19
checkbox attribute, 486
check boxes, 509–511
  creation, 489–490
  identifying, 509
  <input> elements, 509–511
  labels, 510
  use, 491
checked attribute, 490
child elements, **160,** 185, 246
Chrome, 267
  font size samples, 207
class attribute, 26, 167–169, 171–172, 178, 227, 229, 406, 417, 460, 466, 516
  content value, 16
  <img> element, 355
  restricting use, 168–169
class = "chapter" attribute, 433
classes, 460
class names, 25, 457
class selectors, 180, 461
  more specific, 168–169
   elements, 235
  period (.) flag character, 167
  style rules, 167–169
class = "title" attribute, 452–453
clearing elements, 275–276
clear property, 275–276, 302–303, 306, 328
clickable graphics, 420–422
clients, 111
client-side scripting language, 483
closing tags, 20
cluster structure, 132
CMS (content management systems), 113
CNBC Web site, 90
CNET Browser Info Web site, 51
code
  conventions creation, 33
  evaluating, 33
  validating, 51
code-based HTML editors, 29
Code Style Web site, 194
col element, 449
<colgroup> elements, 449, 455–457, 471
  background color, 471–472
collapsing margins, 254
color blindness, 364
color depth, **358,** 384
  GIF (Graphics Interchange Format), 338
color names, 365
color picker, 366
color property, 367–368
colors
  active link, 370
  analogous, 363
  background, 370–371
  basics, 358–367
  <body> element, 369
  borders, 264–266, 368
  browser-safe colors, **359**
  color depth, **358**
  complementary, 360, 362
  controlling properties with CSS, 368–373
  cool, 361
  CSS values, 365–366
  default text color, 369
  dithering, **358**–359
  foreground, 368
  hues, **361**
  inheritance, 369
  JPG (Joint Photographic Experts Group), 341
  links, 370
  meanings, 359–360
  monitors, 10
  monochromatic, 363
  primary, 360
  repeating, 70
  sampling, 366
  secondary, 360
  shades, **361**
  testing, 364
  text, 368, 466
  tints, **361**
  unvisited link, 370
  visited link, 370
  warm, 361
  Web palette, **359**
  wise use of, 364
color schemes, 359–364
  analogous color schemes, **363**
  complementary color schemes, **362**
  monochromatic color schemes, **363**
  types, 362–364
color theory, 360
color values, 365, 368–369
color wheel, 360–361
The Color Wizard Web site, 364
cols attribute, 498
colspan attribute, **451**–452, 475
column drops, **305**–306, 330
column group background color, 456
columns
  background color, 471
  class names, 457
  element width exceeding containing element, 305–306
  grouping, 455–458
  nonfloating, 300–301
  spanning, 451–452
  style characteristics, 456
  width, 455
<command> elements, 23
comments, 157
commercial Web site
  development, 106
  licensing fonts, 195
Common Gateway Interface. *See* CGI (Common Gateway Interface)
common Web fonts, 193–195
complementary colors, 360, 362
complementary color schemes, **362,** 384

complete URLS, **125,** 144
connection speeds
  differences, 51–52
  testing, 141
containing box, **246,** 287
content
  active white space, 76
  backward compatibility, 33
  balancing with design, 66–67
  box model, 248–251
  created by style sheets, 178–179
  design, 108
  development, 110
  easy access to, 67
  flashing visual, 96
  goals, 112–115
  increasing space at box edge, 256
  nontext, 95
  online presentation, 91
  operable, 94, 96
  perceivable, 94–96
  presented in different ways, 95
  robust, 95, 97
  sections, 22, 296–298
  single-sourcing, **18**
  standards, 31
  structuring, 90
  thematic groupings of, 28
  too much, 90
  understandable, 95–97
  users with disabilities, 95–96
content boxes
  automatically adding scroll bar, 277
  background color, 283
  borders, 248–250, 282
  centering text, 283
  controlling overflow, 276–278
  creation, 269
  floating, 273
  left-floating, 274
  margins, 248–250, 281, 283
  minimum and maximum width, 271
  padding, 248–250, 282
  sidebar style, 282–283
  text overflow, 277
  width, 269–271
content containers creation, 296–298
content layer, 367
content management systems. *See* CMS (content managementy systems)
content types
  defining, 19
  HTML (Hypertext Markup Language), 16
context boxes dimensions and position, 268–280
contextual links, 402, **418**–419, 439
contextual selectors. *See* descendant selectors
continuous-tone images, 341
cookies, **29,** 34
cool colors, 361
copying basic.htm file, 157
copyright id, 169
Core Fonts for the Web initiative, 195
Corel Paint Shop Pro, 346
CSS 1, 151
CSS 2, 151
CSS 3, 267–268
CSS3
  browsers, 152
  font-fact property, 195
  selectors, 181–183
  structural pseudo-class selectors, 182–183
  substring matching attribute selectors, 182
  UI element states selectors, 183
  unsupported properties, 222–223
CSS 2.1 attribute selectors, 172
CSS box model, 248–251
CSS box properties, 245
CSS (Cascading Style Sheets), **5**–7, 33–34
  background images, 373–383
  benefits, 151–157
  browser support, 7, 151–152
  color properties, 368–373
  color values, 365–366
  combining style rules with HTML, 153–156
  comments, 157
  display characteristics, 151
  display information, 25
   elements, 170–171
  evolution of, 151–152
  font properties, 202–211
  HTML 4.01, 10–11
  HTML5, 15
  image properties, 353–357
  measurement units, 197–200
  Media Queries, **58**
  properties, 153
  properties cross-reference charts, 152
  separating style information from HTML, 151
   elements, 171–172
  style rules, **152**–156
  styling forms, 500–507
  text spacing properties, 211–221
  Web sites, 13
  writing clean code, 156–157
CSS3 (CSS level 3), 152
CSS handheld media type, 58
CSS level 3. *See* CSS3 (CSS level 3)
CSS style rules, 6, 152
  XML code, 12
CSS visual formatting model, **245**–248, 287
current class, 406
current style rule, 434
cursive fonts, **203,** 237
customized divisions, 170
customizing
  bulleted lists, 230–236
  horizontal navigation bars, 425–426
  marker values, 231
  numbered lists, 230–236
Cute FTP, 138

D

data
  local storage, 29
  single-sourcing, **18**
database administrators, 122
<datalist> elements, 23
declarations, **152**–153, 185
default
  borders, 259
  fonts, 193, 203–205
  main page name, 124
  text color, 369
deprecated elements, 11, 34
descendant elements, 162–163
descendant selectors, 162–163
design, balancing with content, 66–67
designers, 121
designing
  for accessibility, 92–97
  for reading, 81–82
  for scanning, 80, 81–82
designing for users
  flat hierarchy, 87
  hypertext linking effectively, 88–89
  interaction, 80–82
  location, 82–87
  reformatting for online presentation, 91
  too much content, 90
design principles, 49
destination medium and absolute units, 198
<details> elements, 24
development process
  construction, 110
  content development, 110
  graphic design, 108–109
  information design, 108
  maintenance, 111
  page template creation, 108–109
  promotion, 111
  publishing, 111
  quality assurance, 110
  requirements, **107**–108
  specification, 108
  taxonomy, **108**
  user testing, 110
development team, 121–122
devices
  issues, 53–54
  sharing information, 28
  testing work in, 51
Digimarc Web site, 114
digital cameras, 344
digital copyright information, 114
digital watermarking, 114, 345
directory structure, 125, 140
  hierarchical folder structure, 127–128
  single folder structure, 126

display elements, 15
display property, 246–248, 424
  <a> element, 429
display software, 53
display types, testing, 141
dithering, **358**–359, 384
 elements, 25, 170–171, 269, 405
  background color, 372
  style rules, 269
  width, 269
division element, 296–298
divisions, 170–171
doctype (Document Type), **3,** 34
<!DOCTYPE> statement, 15, 18
documentation of style rules, 157
document content type, 15
document language, 212
documents
  body section, 22
  character set, 19
  content type, 19
  converting to HTML, 30
  default font size, 206
  describing structure, 26
  head section, 21
  metadata, 19, **21**
  relationships, 155
  sections, 22, 28, 31
  well-formed, 20
document type, 18
Document Type. *See* doctype (Document Type)
Document Type Definition. *See* DTD (Document Type Definition)
domain names, registering, 137
Dotster Web site, 137
downloading fonts, 205–206
drawing canvas, 28
Dreamweaver, 120
dropcap class name, 176
drop capitals, 175–177
DSL Web hosting service, 136
DTD (Document Type Definition), **18,** 35

E

easy access to information, 67
e-commerce Web sites, 114
electronic files and digital copyright information, 114
elements
  adding padding, 256
  before and after text, 178–179
  attributes, 25
  background color, 372
  block-level, 170, 247
  borders, 258–268, 262
  box shadows, 278–280
  box types, 245
  categories, 21–23
  child, **160**
  class attribute, 167–169
  clearing, 275–276
  closing tags, 20
  descendant, 162–163
  describing document structure, 26
  displaying, 247
  displaying content boxes, 245–248
  embedded content, 22
  empty and closing slash, 20–21
  explicitly setting margins, 255
  first-letter style rules, 175–177
  first line of text, 177–178
  flexible widths, 308
  floating, 272–274, **295**–296
  flow content, 22
  foreground color, 368
  heading content, 22
  height, 272
  hiding, 248
  hierarchical structure, 159–160
  id selector, 169–170
  inheritance, 23
  inline, 22, 170–172, 247
  interactive, 23
  layers, 367
  list type, 247
  lowercase names, 20
  margin settings, 255
  matching meaning and usage of sections, 31
  metadata, **21**
  mixed fixed-width and flexible, 62
  nesting, 160
  new, 23–24
  normal flow, **294**–296, 300–301
  obsolete, 25
  page layout, 27
  page structure, 25–28
  parent, **160**
  phrasing content, 22
  positioning, 272–274
  sectioning content, 22
  sectioning root, 22
  selecting all children of, 163
  selecting by characteristics, 173–180
  selecting on location in document tree, 182–183
  selecting with greater precision, 167
  selectors, 161
  sharing properties, 165
  state of interaction, 183
  styles, 5–7
  top and bottom margins, 254
  transparent, 23
  width, 269–271, 274
e-mail addresses, 136
embedded content elements, 22
Embedded Open Type format, 195
<embed> elements, 24
empty elements, 20–21
em unit, **199,** 207, 237
  navigation button width, 429
enctype attribute, 484
English in Chester Web site, 93
e-readers, 49
European Laboratory for Particle Physics. *See* CERN (European Laboratory for Particle Physics)
expanded fonts, 196
Expression Web, 120
Extensible Hypertext Markup Language. *See* XHTML (Extensible Hypertext Markup Language)
external document fragments, linking, 413–415
external style sheets, 154
  background image URL, 375
  comments, 157
  linking to, 155
  style rules, 155
extranets, **115,** 144
ex unit, **200,** 237
eye dropper tool, 366
eye-tracking studies, 85

F

Facebook, 113
fantasy fonts, **203,** 237
feedback form, 142
fields, grouping, 511
<fieldset> elements, 498–500, 508, 511
  background image, 506–507
  styling, 504–507
  width, 505
fieldsets
  forms, 517–518
  white space, 518
:figtitle class name, 179
<figure> elements, 24, 27
file attribute, 486
file extensions, 124
file input type object, 493–494
filename conventions, 122–125
files
  naming, 123–124
  uploading with FTP (File Transfer Protocol), 138–140
  users submitting, 493–494
File Transfer Protocol. *See* FTP (File Transfer Protocol)
Filezilla, 138
Firebug, 121
Firefox, 54
  border styles, 260
  fonts in default browser size, 196
  generic font families, 203
  support for CSS, 7
:first-letter pseudo-element, 175–177
first line of text, 177–178
:first:line pseudo-element, 177–178
fixed_layout.htm file, 319
fixed_layout1.htm file, 319
fixed layouts, **316,** 330
  article division, 325–326
  article element, 317
  centering, 318–319
  creation, 319–329

floating elements, 328–329
footer division, 328–329
header division, 322–323
header element, 317
nav division, 324
nav element, 317
pixel measurements, 316
sidebar division, 327
style rules, 317
wrapper division, 317–319, 321
wrapper element, 316
fixed-size elements, 316
fixed-width fonts, 203
fixed-width page layouts, **63**–65, 99
advantages, 66
reading, 83
Flash, 340
flashing visual content, 96
flat hierarchy, 87
flexible elements style rules, 308
flexible_layout.htm file, 311
flexible_layout1.htm file, 311
flexible layouts, **59**–62, 99, **307**–309, 330
adapting to browser size, 308
advantages, 66
article division, 314
controlling width, 310
creation, 310–316
header division, 312
nav division, 313
reading, 83
screen resolutions, 308
floating, 330
content box, 273, 282
labels, 502, 515
tables, 463–465
floating elements, **295**–296
column drops, **305**–306
containing, 299
controlling flow of text around, 275–276
within floating elements, 303–304
order, 303–304
pull quote, 303
style rules, 308
width, 299
floating images, 273, 355
floating layouts, 298–299
clearing problem, 306
clear property, 302–303
floating elements within floats, 303–304
normal flow element, 300–301
float property, 176, 272–274, 355, 428–429
flow content elements, 22
fluid layouts. *See* flexible page layouts
focus, 111
font activity.html file, 224
font activity1.html file, 224
font and text properties style sheet, 224
adding <style> section, 225
capitalizing key words, 228–229
second-level headings, 227–228
styling headings, 225
styling paragraphs, 226–227
font-face property, 195, 205–206
font families, 194, 202–205
captions, 472–473
forms, 508
generic, 203
name containing white space, 204
specific, 204–205
font-family property, 159, 202–205, 225–226, 281
font properties, 202–209, **210**–211, 237
abbreviating property listings, 211
basic use of, 210
font-face property, 205–206
font-family property, 202–205
font property, **210**–211
font-stretch property, 210
font-style property, 208
font-variant property, 208–209
font-weight property, 209
line-height, 211
fonts, **192**
absolute size keywords, 206
alternate, 205
common Web, 193–195
consistency, 193
Core Fonts for the Web initiative, 195
cursive, **203**
default, 193, 203, 205
design for legibility, 196–197
downloading, 205–206
em unit, 199
expanding or compressing, 196, 210, 223
ex unit, **200**
fallbacks, 205
fantasy, **203**
fixed-width, 203
font families, 202–205
generic names, 203
Microsoft EOT format, 205
monospace, **203**
proprietary, 195
purchased and licensed, 206
relative sizes, 207
repeating, 70
sans-serif, 194, **203**
serif, 194, **203**
setting multiple properties, 210–211
sizes, 199, 206–207, 223
specialty, 197
styles, 208
TFF (TrueType Format), 205
unavailable, 203
user-controlled sizes, 93
using fewer, 192–193
Web sites licensing, 195
x-height, **196**
font-size-adjust property, **223**
Font Squirrel Web site, 195
font-stretch property, 210, **223**
font-style property, 208
font-variant property, 208–209
font-weight property, 209, 229
footer division fixed layouts, 328–329
<footer> elements, 24, 27, 302–303
footers and tables, 454–455
footer style rule, 328
for attribute, 488, 513
foreground color, 368
formalizing testing, 142
formatting
article division, 314
header division, 312
nav division, 313
form controls, **484,** 520
<form> elements
attributes, 484
clickable area, 487–488
form controls, **484**
mailto action, 485
structural elements, 484
form.html file, 507
forms
aligning elements, 501–503
background color, 505
building, 507–519
check boxes, 489–490, 509–511
collecting data, 485
creation, 484–485
entering data, 483
fieldsets, 517–518
font family, 508
form controls, 484
get, 484–485
grouping fields, 511
grouping input elements, 498–500
input objects, 485–500
JavaScript, **483**
labeling elements, 486–488
labels, 502, 508
label styles, 515–517
legends, 517–518
list boxes, 494–497, 512–513
mailto action, 485
margins, 508
operation, 482–483
password entry field, 494
post, 484–485
processing, 483
radio buttons, 490–491, 512–513
reset button, 491–494, 503, 514
scrollable list of options, 494–497
sending data to e-mail address, 485
sending data to server, 484–485
styling, 500–507
submit button, 491–494, 503, 514
text area, 497–498
text boxes, 488
text for legend, 506
fragment identifiers, **409,** 410–412, 416, 439
framesets, 15
freeware, **120**–121, 144

freeware graphic-editing programs, 346
freeware wireframe software tools, 110
F-shaped pattern, 84
FTP clients, 138
FTP (File Transfer Protocol), **138**–140, 144
Fugu, 138
full-featured HTML editors, 29

G

gateway routers, 136
generic font families, 203
get, 484–485
GIF Construction Set, 340
.gif extension, 124, 340
GIF (Graphics Interchange Format), 124, **338**–340, 384
  advantages and disadvantages, 345
  animation, 339–340
  banding, **359**
  color depth, 338
  interlaced, 343
  lossless compression, 338
  transparent colors, 339
GIFMation, 340
Gliffy Web site, 110
global attributes, 25
goals, 111–115
Go Daddy Web site, 137
Google Web site, 78
  site map, 87
graphics
  See also images
  advantages and disadvantages, 345
  aligning text with, 217
  alternative text-based links, 420–421
  consistency, 420
  designing, 108–109
  file formats, 338–343
  GIF (Graphics Interchange Format), **338**–340
  icons for navigation, 421–422
  indicating history, 432–433
  indicating location, 433–434
  interlacing, 343–344
  irregular outside shape, 339
  JPG (Joint Photographic Experts Group), **341**–342
  linking, 420–422
  minimizing download time, 420
  navigation, 400, 420–422, 432–434
  PNG (Portable Network Graphics), **342**
  predictable navigation cues, 420
  professional-quality, 346
  progressive display, 343–344
  quickly downloading, 352
  reusing, 53
  sizing for Web page, 352–353
  SVG (Scalable Vector Graphics), **342**–343
  text as, 197
graphics editors
  saving images in interlaced (progressive) format, 343
  transparent areas, 339
Graphics Interchange Format. *See* GIF (Graphics Interchange Format)
graphics tools, 345–346
  shareware or freeware, 346
grids, **72**–74, 99
grouping
  columns, 455–458
  fields, 511
  input elements, 498–500
  list options, 496–497
  radio buttons, 490–491, 513
  rows, 454–455
grouping selectors, 162
Guardian site main page, 74

H

handheld devices, 49
  screen resolution, 57–58
hanging indent, 212
<head> element, 4
  <link> element, 154, **155**
  <style> element, 156, 158, 164, 225
<head> elements, 160
header division
  fixed layouts, 322–323
  formatting, 312
  margins, 312
<header> elements, 24, 27, 317–318
headers in tables, 449–450, 454–455
header style rule, 322
heading elements, 22
headings and style rules, 225
height attribute, 346, 349–352
height property, 272, 308, 318
<h1> elements, 246
  background color, 373
  background color and padding, 371
  margins, 281
  style rules, 158, 164–165, 225
<h2> elements
  style rules, 164–165
  styles, 227–228
<h3> elements style rules, 164
hexadecimal color values, 365–366
hexadecimal numbers, 366
hidden attribute, 486
hiding elements, 248
hierarchical folder structure, 127–128
hierarchical structure, 131
high-contrast version, 93
Home link graphic, 86
horizontal navigation bars
  background color, 426
  basic, 425–426
  borders, 426
  button width, 428–429
  customizing, 425–426
  display property, 424
  margins, 426
  padding, 426
  text decoration, 426
  width, 427
horizontal navigation lists, 247
horizontal repeating background graphic, 378–379
Horton, Sarah, 85
hover
  changing background color, 436–437
  changing text color and background color, 435
  underlining, 437–438
hover effects
  background, 467–468
  background color, 473
  style rules, 436
:hover pseudo-class, 174–175, 435, 438, 468
hover rollovers, 435–438
href attributes, 126, 406, 409, 411, 414
h2 selector, 227
hspace attribute, 347–348
HTML 4.01, 10–11
  display elements, 11
HTML5
  animations, 14
  attributes, 21, 25
  backward compatibility with HTML 4.01, 16
  choosing correct syntax, 17–18
  closing tags, 20
  compatibility, 15
  CSS (Cascading Style Sheets), 15
  display elements, 15
  <!DOCTYPE> statement, 15, 18
  element categories, 21–23
  element names, 20
  elements designed for rich media, 14
  empty elements, 20–21
  features removed from, 15
  framesets and frames, 15
  interactive capabilities, 28–29
  logical layout elements, 14
  loose and strict syntaxes, 16–17
  <meta> element, 15
  migrating from legacy HTML to, 32–33
  MIME (Multipurpose Internet Mail Extensions), **19**
  new elements, 23–24
  new form input types, 486
  obsolete elements, 25
  page structure elements, 25–28
  proposal for, 13–14
  tags, 20
  well-formed document, **20**
  XHTML MIME type, 16
  XHTML Namespace qualifier, 16
  XHTML version of, 16–17
  XML syntax rules, 16
HTML code, 3, 12
  character set, 15
  document content type, 15
  good practices, 30–33
  semantic markup, **31**

standards, 31
syntactically correct, 19–21
testing on browsers, 50
validating, 32, 51
HTML developers, 121
HTML development tools, 120–121
HTML documents, 3–4
separation of style information, 11
structure, 159–160
HTML editors, 29–30
link validation tools, 141
validators, 32
<html> elements, 160
.html extension, 124
HTML (Hypertext Markup Language), **2,** 9, 35
browsers, 5
combining CSS style rules with, 153–156
content type, 16
deprecated elements, 11
first version, 9
hexadecimal numbers, 366
history of, 8–14
hyperlinks, 400
interoperability, 11
migrating to HTML5, 32–33
misuse of elements, 4
new coding techniques, 4
problems with, 10
real-world coding practices, 8
releases, 11
standards for, 10
syntax, 12
table-based designs, 9
viewing source code, 4
Web page creation, 2–7
HTML tables, 10
HTTP (Hypertext Transfer Protocol), 135, 483
hues, **361,** 363, 384
hyperlinks, 400
changing style characteristics, 173–180
clickable, 8
hovering style, 174
hypertext, **8,** 35
connecting facts, relationships, and concepts, 402
defining, 400
document navigation, 400
effectively linking, 88–89
organizing information, 8
removing borders from image, 353
hypertext-linked content, 400
hypertext links, 8
disabling default underlining, 437
underlined text, 219–220
Hypertext Markup Language. *See* HTML (Hypertext Markup Language)
Hypertext Transfer Protocol. *See* HTTP (Hypertext Transfer Protocol)

I

ICANN (Internet Corporation for Assigned Names and Numbers), 137
icons, 421–422
id attribute, 26, 169–171, 405, 409, 411, 416, 488, 513
id names, 25–26
id selector, 169–170, 180, 405
IE (Internet Explorer) 5 for Macintosh, 151
IIS (Internet Information Service), 483
image attribute, 486
images, 27
See also graphics
acquiring, 344–345
aligning, 354
aligning text along borders, 354
alternate text, 350
aspect ratio, **351**
borders, 348
borrowing from Web sites, 345
bulleted lists, 233
continuous-tone, 341
controlling properties with CSS, 353–357
copyright or identifying information, 348
description, 348
digital cameras, 344
digital watermarking, 345
dithering, 358–359
floating, 348, 355
height, 349–352
within line of text, 347
location of file, 492
margins, 348
multiple, 339–340
normal alignment, 346–347
numbered lists, 233
offsetting text from, 273
positioning, 272–274
public domain Web sites, 344
removing hypertext border from, 353
scanners, 344
space required by, 349
stock photo collections, 344
submit button, 492–493
tiling, 375–377
timing information, 339–340
user-created, 345
white space, 348, 356–357
width, 349–352
<img> elements
alt attribute, 346, 420–421
class attribute, 355
floating, 303
height attribute, 346
replacing attributes with style sheet properties, 347–348
src attribute, 346–347
style attribute, 217
title attribute, 346
width attribute, 346
!important keyword, **181,** 184
indenting text, 212–213
index.html file, 124, 403
Index page, 403
indicating location with background color or graphics, 433–434
information
catalog structure, 133–134
clear presentation, 68
cluster structure, 132
design, 108
easy access to, 67
hierarchy of, 70, 131
linear structure, 129
organizing structure, 129–134
tutorial structure, 129–130
Web structure, 131
information designers, 122
information overload, 402
inheritance, **160,** 185
color, 369
indenting text, 212
writing simpler style rules, 159–161
initial capitals, 175–177
initial class name, 175–176
inline elements, 22, 170–172, 247
naming, 171–172
normal flow, 295
vertical-align property, 216
inline-level boxes, 245
inline list elements, 424
inline style, 154, 156, 180
inline text and background image color, 371
inline text boxes, 246
<input> elements, 508, 510, 513
captions, 486–488
grouping, 498–500
id attribute, 488
types, 485–500
interactions, 80–82
interactive capabilities, 28–29
interactive elements, 23
interlacing, 343–344, 385
PNG (Portable Network Graphics), **342**
internal linking, 409–410
internal style sheets, 154, 156
Internet Browser Review Web site, 51
Internet Corporation for Assigned Names and Numbers. *See* ICANN (Internet Corporation for Assigned Names and Numbers)
Internet Explorer
border styles, 260
Embedded Open Type format, 195
fonts in default browser size, 196
generic font families, 203
support for CSS, 7
Internet Explorer 8
proprietary Microsoft font format, 205
text-shadow property, 221

Internet Explorer 9 and SVG graphics, 343
Internet Information Service. *See* IIS (Internet Information Service)
Internet service providers. *See* ISPs (Internet service providers)
intranets, **115,** 117, 144
intro class selector, 167–168
iPhone, 57–58
ISO 9660 Standard, **123,** 144
ISPs (Internet service providers), **136,** 144
italic text, 208

J

JavaScript, **483,** 520
Joint Photographic Experts Group. *See* JPG or JPEG (Joint Photographic Experts Group)
.jpeg extension, 124
JPEG (Joint Photographic Experts Group), **341**–342, 385
.jpg extension, 124
JPG (Joint Photographic Experts Group), **341**–342, 385
  advantages and disadvantages, 345
  color, 341
  dithering files, 359
  files, 124
  lossy compression, 341
  progressive format, 343
  twenty24-bit color display, 358
justified text, 223

K

kerning, 218
keyboard, 96
keywords
  border-style property, 259
  capitalizing, 228–229
Kompozer, 121

L

label attribute, 496–497
label class, 515
<label> elements, 486–488, 491, 502, 508, 510
labels
  check boxes, 510
  floating, 502, 515
  forms, 486–488, 502, 508
  margins, 503
  right-aligning text, 503
  styles, 515–517
lang attribute, 212
languages
  alignment values, 214
  indenting text, 212
LAN (local area network), 115
large keyword, 206
leading, **214,** 237
left-floating content box style rules, 274
left margin, 227
<legend> elements, 498–500, 504–508
legends
  forms, 517–518
  text, 506
length value, 251, 269, 381
Let's Go Web site, 68
letter spacing, 217–218, 227–228
letter-spacing property, 217–218
licensed fonts, 206
Lie, Hakon, 207
 elements, 422, 428–430
linear structure, 129
line height, 214–215
line-height property, 214–215, 226
LinkedIn, 113
<link> elements, 154–**155,** 185
linking
  chapter pages, 407–408
  contextual, 402
  external document fragments, 413–415
  graphics, 420–422
  hypertext effectively, 88–89
  internal, 409–410
  text-based, 402–420
  text navigation bar, 404–407
:link pseudo-classes, 173–174, 370, 432
links, 8, 88–89
  alternate text-based, 420–421
  avoiding extra space in, 408
  colors, 370
  contextual, 418–419
  interaction, 80
  moving through content structure, 402
  page turners, 416–418
  table of contents, 403
  testing, 141
link validation tools, 141
Linux common fonts, 194
Linux Web servers, 483
liquid layouts. *See* flexible layouts
list boxes, 494–497
  forms, 512–513
list elements, 424
list-item value, 230
lists
  bulleted, 230–236, 422
  default value, 495
  grouping options, 496–497
  horizontal items, 247
  items visible in, 496
  markers placement, 234–235
  multiple values, 495–496
  navigation, 422–423
  numbered, 230–236
list-style-image property, 233, 236
list-style-position property, 234–236
list-style properties, 230–231, 233–236
list-style shorthand property, 235–236
list-style-type property, 231, 236, 247, 423
list type elements, 247
Literary Machines (Nelson), 400
local area network. *See* LAN (local area network)
local data storage, 29
locating user, 401–402
logical divisions, 170–171
logo inline element, 171
look and feel, **66**–68, 99
Los Angeles Zoo Web site, 70–71
ltbluebtn.jpg file, 437
Lynch, Patrick J., 85

M

Macintosh
  common fonts, 194
  default fonts, 193
  TextEdit, 29
mailto action, 485
main division, 297
maintenance, 111
manageable information segments, 402
MapQuest Web site, 421
margin-bottom property, 252
margin-left property, 251
margin properties, 249, **251**–255, 274, 287, 357, 463–464
  automatic centering, 318
   element, 423
margin-right property, 251
margins, 74, 226, 251–255
  collapsing, 254
  content boxes, 248–251, 283
  em unit, **199**
  explicitly setting, 255
  forms, 508
  header division, 312
  horizontal navigation bars, 426
  images, 348
  labels, 503
  left and right, 252
  length value, 251
  negative, 253
  percentage value, 251
  pixel values, 251
  removing default in lists, 423
  removing default spacing, 254–255
  specifying, 251–252
  style rules, 251
  tables, 463–465
  text, 284
  transparent, 248
  values, 274
  vertical, 254
  vertical navigation bar, 430
  white space, 518
  zeroing, 254–255
margin-top property, 252
<mark> elements, 24
marker values, 231
markup languages, **2,** 35
max-height property, 272
maxlength attribute, 488
max-width property, 271, 310

measurement units, 197–200
  absolute units, 198
  destination medium, 197
  em unit, **199,** 207
  ex unit, **200**
  percentages, 199, 207
  pixels, 207
  px unit, 200
  relative units, 199–200
measurement values, 251
Media Queries, **58,** 99
medium keyword, 206, 262
metadata, **21,** 35
<meta> elements, 4, 19
<meter> elements, 24
method attribute, 484
Microsoft Core Fonts for the Web initiative, 195
Microsoft EOT format, 205
Microsoft Expression Web, 29, 138
Microsoft Internet Explorer, 54
Microsoft PowerPoint, 30
Microsoft Project, 106
Microsoft WOFF (Web Open Font Format), 195
Microsoft Word, 30
MIME (Multipurpose Internet Mail Extensions), 16, **19,** 35
min-height property, 272
min-width property, 271, 310
mission statement, 111
mockups, 108–109
monitors, 53, 358
  colors, 10
  lower-resolution, 321
  resolution, 10
  screen resolution, **54–66**
  wide-screen displays, 55
monochromatic color schemes, **363,** 385
monospace fonts, 195, **203,** 237
Mozilla and WOFF (Web Open Font Format), 195
multimedia, synchronized alternatives for, 95
multiple attribute, 495–496
Multipurpose Internet Mail Extensions. *See* MIME (Multipurpose Internet Mail Extensions)
MySpace, 113

N

name attribute, 409, 411, 413, 489–491, 508–509, 513
naming
  files, 123–124
  style rules, 167–169
NASA home page, 76
National Archives Web site, 401
National Public Radio News Web site. *See* NPR (National Public Radio) News Web site
nav division, 318
  fixed layouts, 324
  formatting, 313
<nav> elements, 24, 27, 308
  fixed layouts, 317
  floating, 301
navigation
  background color, 432–434
  breadcrumb path, **402**
  button width, 429
  cues to location, 401
  graphics, 400, 420–422, 432–434
  hypertext documents, 400
  hypertext-linked content, 400
  icons, 421–422
  indicating history, 432–433
  indicating location, 433–434
  limiting information overload, 402
  links, 93
  lists, 422–423
  locating user, 401–402
  paper-based media, 400
  repeating options, 420
  text links, 400, 402–420
  usable, 400–402
navigation bars, 247
  building, 404–407
  buttons changing colors, 436–437
  chapter files, 416
  horizontal, 424–429
  style rules, 435
  vertical, 429–431
navigation division, 297
navigation elements, 70
navigation questions, 401
navigation structures
  building, 402–420
  chapter pages links, 407–408
  contextual links, **418**–419
  external document fragments links, 413–415
  internal linking, 409–410
  page navigation bar, 410–412
  page turners, 416–418
  summary, 419–420
  text navigation bar linking, 404–407
negative margins, 253
negative shadow style rules, 279
Nelson, Ted, 400
nesting elements, 160
NetMarketShareStats Web site, 55
Netmarketshare Web site, 51
Netscape, 483
Newton, Isaac, 360
The New Yorker Web site, 79
New York Times, 90
Nielsen, Jakob, 84
960 grid system standard, 75
noncommercial Web sites, 17
none value, 248
nonrepeating background image, 379–380
nontext content text alternatives, 95
normal flow, **294**–296, 331
normal flow elements, 300–301
note class attribute, 208
Notepad, 29
NPR (National Public Radio) News Web site, 80–81
numbered lists
  customizing, 230–236
  images, 233
  list-item properties, 235–236
  list markers placement, 234–235
  marker values, 230–232
numbering system, 231

O

objects
  floating text around, 348
  types, 485–500
oblique text, 208
odd class, 466
Office applications, converting documents to HTML, 30
 elements, 230, 233, 235
online feedback form, 116
online shopping Web sites, 114
Open Font Library Web site, 195
Open Type format, 195
Opera, 54
  border styles, 260
  box-shadow property, 278–280
  fonts in default browser size, 196
  generic font families, 203
  support for CSS, 7
  WOFF (Web Open Font Format), 195
operable content, 94, 96
operating systems, 53–54
  fonts, 193
  testing, 140
<optgroup> elements, 496–497
optional navigation links, 93
<option> elements, 494–495, 512
organizing information
  hypertext, 8
  structure, 129–134
<output> elements, 24
overflow property, 277–278
overriding
  styles, 156
  universal selector, 163
oz.htm file, 163
oz1.htm file, 163

P

padding, 256, 282
  cells, 470
  content box, 248–251
  em unit, **199**
  horizontal navigation bars, 426
  individual areas settings, 256
  pixel values, 251
  removing default in lists, 423
  tables, 461–463
  vertical navigation bar, 430–431

padding area, 248, 255–257
padding-bottom property, 165, 257
padding-left property, 256
padding property, 249, **255**–257, 287, 423, 461–462
  measurement values, 251
  shorthand notation, 257
padding-right property, 257
padding-top property, 257
page layout box properties, **268**–280, 287
  box shadows, 278–280
  clearing elements, 275–276
  controlling overflow, 276–278
  element height, 272
  element width, 269–271
  floating elements, 272–274
page layout creation, 280–286
page layout elements, 27
page navigation bar, 410–412
pages
  background color, 372–373
  backgrounds, 375–376
  structure elements, 25–28
page turners, 416–418
page views, **118,** 144
Paint Shop Pro, 346
paletteman Web site, 364
paper-based media, 400
paragraph element
  box model, 248
  left and right margins, 252
paragraphs
  background color, 249
  box property settings, 250
  style rules, 226–227, 249, 250
  vertical margin between, 254
  white space, 222
  wrapping text, 222
parent elements, **160,** 185, 246
partial URLS, **125**–126, 144
passive white space, **75,** 99
password attribute, 486
password entry field, 494
password input type object, 494
PCs, 194–195
<p> elements, 226, 246, 405, 409, 416, 502, 510, 513
  class attribute, 227
  id attribute, 411
  no closing tag for, 16
  style rules, 159, 165–166
Pencil Web site, 110
people with disabilities and content, 95–96
perceivable content, 94–96
percentage, 207
percentage elements, 316
percentage value, 381
  margins, 251
  width property, 269
period (.) flag character, 167
perspective, 112
photographs, 341
  dithering, 359
  JPG format, 344
  saving original copy, 342
Photoshop Save for Web & Devices dialog box, 341–342
.php extension, 124
pixels, 200, 207
pixel values, 251
planning Web sites
  analyzing audience, 115–121
  content goal, 112–115
  directory structure, 125–128
  filenames and URLs conventions, 122–125
  site specification, **111**–112
  storyboards, 128–134
  Web site development team, 121–122
platforms
  browsers, 51
  sharing information, 28
  users, 117
.png extension, 124
PNG (Portable Network Graphics), **342,** 385
  advantages and disadvantages, 345
  files, 124
  interlacing, **342**–343
  twenty24-bit color display, 358
Poe, Edgar Allen, 91
polyglot document, 17
Portable Network Graphic. *See* PNG (Portable Network Graphic)
portals, 113
post, 484–485
pound sign (#) flag character, 169
presentation information, 33
primary colors, 360
printing
  absolute units, 198
  italic or oblique text, 208
procedure style, 168–169
product support Web sites, 115
professional Web sites, 17
<progress> elements, 24
progressive display, 343–344
project development process, 106–111
project.html file, 469
project management team, 121
project plan, 106
promotion, 111
properties, **153,** 185
  colors, 368–383
  combining declarations, 162
  descriptions, 201
  images, 353–357
  inheritance, 160
  names, 153
  values, 153
proprietary animation tools, 340
pseudo-classes, **173**–180, 185
pseudo-elements, **173**–180, 185
public domain Web sites, 344
publishing
  registering domain name, 137
  Web sites, 111
publishing industry repurposing content, 17–18
publishing Web sites, 113
  uploading files with FTP (File Transfer Protocol), 138–140
  Web hosting service comparison checklist, 138
  Web hosting service provider, 135–137
pull quote, 303
px unit, 200

Q

quality assurance, 110

R

radio attribute, 486
radio buttons, 490–491, 512–513
raster graphics, **343,** 385
reading
  designing for, 81–82
  eye-tracking studies, 85
  fixed-width page layout, 83
  flexible page layout, 83
  in F-shaped pattern, 84
Readprint Web site, 91
Real Simple Syndication. *See* RSS (Real Simple Syndication)
recommendations, **152,** 185
Register.com Web site, 137
registering domain name, 137
Rehabilitation Act Amendments of 1998, 94
relative file structure, 127
relative measurement values, 251
relative paths, 126
relative units, 199–200
relative values, 262
rel attribute, 155
rendering engine, **5,** 35, 50
requirements, **107**–108, 112, 144
reset attribute, 486
reset button, 491–494, 503, 514
resolution, 197
  monitors, 10
  pixels, 200
reusing graphics, 53
reverse text, 373
RGB (red, green, and blue) colors, 358, 366
robust content, 95, 97
root directory and partial URLS, 126
<root> elements, **3**–4, 35
rounded borders, 267–268
routers, 136
rows, 449–450
  alternate color, 466
  grouping, 454–455
  padding properties, 462

spanning, 452
styling borders, 460
rows attribute, 498
rowspan attribute, **452**–453, 475
<rp> elements, 24
RSS (Real Simple Syndication), 114
<rt> elements, 24
<ruby> elements, 24

S

Safari, 54, 57
border styles, 260
fonts in default browser size, 196
generic font families, 203
support for CSS, 7
sampling colors, 366
sans-serif fonts, 159, 194, **203,** 237
Scalable Vector Graphics. *See* SVG (Scalable Vector Graphics)
scanners, 344
scanning, designing for, 80, 81–82
screen resolution, 49, **54,** 99, 197
adapting to lower, 62
designing for multiple, 54–66
fixed-width page layouts, **63**–65
flexible page layouts, **59**–62
handheld devices, 57–58
suggestions for solving, 65–66
wide-screen displays, 55
ScreenTips and title text, 421
scripts, **483,** 520
scroll bars
automatically adding to content box, 277
text area, 498
SeaMonkey, 121
search engines
primary source of information for, 4
term entered to find site, 117
secondary colors, 360
<section> elements, 24, 27–28
sectioning content, 22
sectioning root, 22
sections, 22, 31
content, 296–298
heading structure, 28
naming conventions, 26
Secure Sockets Layer. *See* SSL (Secure Sockets Layer)
security with DSL and cable, 136
seizures and flashing visual content, 96
selected attribute, 495
<select> elements, 494–497, 512
selection techniques
applying, 163–166
class and ID selectors, 166–170
combining declarations, 162
descendant selectors, 162–163
grouping selectors, 162
type selectors, 161–162
universal selector, **163**
selectors, **152,** 161, 185
attribute, 172
combining property declarations, 162
containing multiple elements, 163
CSS3, 181–183
descendant, 162–163
grouping, 162
pseudo-class, 173–180
pseudo-element, 173–180
separating with commas, 162
specificity, 180
structural pseudo-class, 182–183
substring matching attribute selectors, 182
UI element states, 183
universal, 163
semantic markup, **31,** 35
Semantic Web Activity group, 31
serif fonts, 194, **203,** 237
server administrators, 122
server logs, 117
SGML (Standard Generalized Markup Language), **9,** 35
shades, **361,** 363, 385
shareware, **120,** 144–145
graphic-editing programs, 346
Shopping cart element, 86
shorthand notation
border-color property, 265–266
border-style property, 261
border-width property, 264
margin properties, 252
padding property, 257
shorthand properties and borders, 266–267
sidebar division in fixed layouts, 327
sidebar style, 282–283
sidebar style rule, 327
single folder structure, 126
single-sourcing, **18,** 35
site map, 87
sitemap.html file, 403
sites. *See* Web sites
site specification, **111**–112, 145
size attribute, 488, 496, 508
small capitals, 208–209
small keyword, 206
smooth transitions, 70–71
social networking Web sites, 113
software tools, 120–121
source code, 4
<source> elements, 24
span attribute, 455–457
 elements, 171–172, 228, 371, 434
class attribute, 229, 406, 417
small capitals, 209
spanning
columns, 451–452
rows, 452–453
special class selectors, 167
Special interest Web sites, 113
specialty fonts, 197
specification, 108
specific font families, 204–205
specificity weight for style rules, 180
SQL Database support, 136
SQL (Structured Query Language), **136,** 145
src attribute, 126, 172, 346–347, 492
SSL (Secure Sockets Layer), **137,** 144
standards
HTML code, 31
W3C, 51
Web and, 9–11
standards-compliant browsers, 51
State of Maine Web site, 116
stock photo collections, 344
storyboards
creation, 128–134
organizing information structure, 129–134
 elements, 209
structural elements, 484
structural pseudo-class selectors, 182–183
Structured Query Language. *See* SQL (Structured Query Language)
style attribute, 156, 233, 353
<style> elements, 6, 156–158, 164, 225, 312
style rules, **152**–156, 158, 185, 225
basic formatting properties, 311–312
borders, 459–460
box-shadow paragraph, 279
cascade effect, 180–181
class name, 175
class selectors, 167–169
clean CSS code, 156–157
combining declarations, 162
combining with HTML, 153–156
declarations, **152**–153
descendant selectors, 162–163
 elements, 170–171, 269
documentation, 157
external style sheets, 155
:first-letter pseudo-element, 175–177
:first-line pseudo-element, 177–178
fixed layouts, 317
flexibility in application of, 154
floating elements, 308
floating images, 273
footer element, 302
grouping selectors, 162
<h1> element, 158, 164–165, 225
<h2> element, 164–165
<h3> element, 164
hover effect, 436
:hover pseudo-class, 174–175
id selector, 169–170
!important keyword, **181**
inheritance, 159–161
internal style sheet, 156
left-floating content box, 274
margins, 251
naming, 167–169
negative shadow, 279

order weight, 181
paragraphs, 249, 250
<p> elements, 159, 165–166
precedence, 180–181
selecting elements by characteristics, 173–180
selectors, **152**
sources, 154
 elements, 171–172
specificity weight for, 180
specific selectors, 180
type selectors, **162**
universal selector, **163**
styles, 5–7
borders, 259–261
class selectors, 180
CSS rules, 6
elements, 5–7
id selectors, 180
labels, 515–517
overriding, 156
separation in HTML documents, 11
tables, 469–474
styles.css file, 154, 404, 406, 417–418
style sheets, **6,** 35, **152,** 185
:after pseudo-elements, 178–179
basic selection techniques, 163–166
:before pseudo-elements, 178–179
box properties, 285–286
building basic, 157–159
cascade, **180**–181
content created by, 178–179
external, 155
internal, 156
relative URL of, 155
style rule based on order in, 181
styling
borders, 458–461
captions, 457–458
<fieldset> element, 504–507
forms, 500–507
<legend> element, 504–507
table borders, 458–461
submit attribute, 486
submit button, 491–494, 503, 514
submit class, 503
subscript, 216
substring matching attribute selectors selectors, 182
superscript, 216
superscript class, 216
SVG (Scalable Vector Graphics), **342**–343, 385
advantages and disadvantages, 345
symbols, 231
synchronized alternatives for multimedia, 95

T

table elements, 448–453
<table> elements, **449**–450, 475
table model values, 247
table of contents
chapter links, 413
fragment identifiers, 410–412
links, 403
Table of Contents page, 403
Back to Top link, 409
internal linking, 409–410
linking to other main pages, 404
links to chapter files, 407–408
table-row value, 247
tables
alternate color rows, 466
background colors, 465–468
background hover effects, 467–468
basic, 449–450
borders, 470
box properties, 461
captions, 450, 471
cells, 449, 450
collapsing borders, 451
data cells, 459
floats, 463–465
footers, 454–455
grouping columns, 455–458
header cells, 459
headers, 449–450, 454–455
information about, 449
interactivity, 467
margins, 463–465
padding, 461–463
rows, 449–450
spanning columns, 451–452
spanning rows, 452–453
styles, 469–474
styling borders, 458–461
styling captions, 457–458
white space, 463–464
tags, 20
target audience, 112
taxonomy, **108,** 145
<tbody> elements, 449, 454–455
<td> elements, 448–**449,** 459, 461–462, 466, 475
technology issues for audience, 119–120
The Tell-Tale Heart (Edgar Allen Poe), 91
template files, 19
templates
creation, 108–109
Web hosting service, 135
testing Web sites, 140–142
text
aligning, 213–214, 217, 354
appearance, 202–211
capitalization, 220–221
centering in cells, 452
changing color on hover, 435–438
colors, 368, 466
complementary colors, 362
controlling flow around, 275
default color, 369
first line of, 177–178
floating elements, 275–276, 348
font size, 199
as graphics, 197
images in, 347
indenting, 212–213
interactions, 81
italic, 208
legends, 506
legibility, 197, 215, 223
letter spacing, 217–218
line height, 214–215
margins, 284
oblique, 208
offsetting from image, 273
overflow in content box, 277
percentage values, 199
readable and understandable, 97
reverse, 373
shadows, 221
subscript and superscript, 216
text decorations, 219–220
underlining, 219–220
vertical alignment, 215–217
white space, 197, 215
words broken, 223
word spacing, 218–219
wrapping, 222
text-align-last property, **223**
text-align property, 213–214, 452–453
text area, 497–498
<textarea> elements, 497–498
text attribute, 486
text-based linking, 402–420
text boxes, 488
inline, 246
text-decoration property, 219–220, 437–438
text decorations, 219–220, 426
TextEdit, 29
text-emphasis property, **223**
text-indent property, 212–213
text links, 400
text navigation bar, linking, 404–407
text-outline property, **223**
text property, 237
text-shadow property, 221
text spacing properties
left, center, right, and justify values, 213–214
letter-spacing property, 217–218
line-height property, 214–215
text-align property, 213–214
text-decoration property, 219–220
text-indent property, 212–213
text-shadow property, 221
text-transform property, 220–221
vertical-align property, 215–217
word-spacing property, 218–219
text-transform property, 220–221, 228
text-wrap property, **222**
TFF (TrueType Format), 195, 205
<tfoot> element, 449, 454–455
<thead> element, 448
<thead> elements, 454–455

<th> elements, 448–**449,** 450, 459, 471, 475
background color, 461, 466
border, 461
padding property, 461–462
text color, 466
themes, 69
thick keyword, 262
thin keyword, 262
three clicks rule, 87
tiling background images, 374
tiling images, 375–377
<time> elements, 24
tints, **361,** 363, 385
title attribute, 172, 346, 348, 421
<title>elements, 4
titles
HTML documents, 4
text, 421
toc.html file, 403–404, 407–413, 415
ToolTips and title text, 421
topic headings, 413
topic links, 414–415
transitions, 70–71
transparency and PNG (Portable Network Graphic) files, 342
transparent colors, 339
transparent elements, 23
<tr> elements, 448, **449**–450, 466, 475
Trellian, 121
TrueType Format. *See* TFF (TrueType Format)
tutorial structure, 129–130
type attribute, 485
type design principles, 192–197
typefaces, **192**–193, 238
Typekit Web site, 195
type selectors, 161–**162,** 180, 185
typography, 192
Typotheque Web site, 195

U

UI element, 183
Ulead PhotoImpact, 346
 elements, 230, 422–423, 426–427
underlining
hover, 437–438
text, 219–220
understandable content, 95–97
unified design
active white space, **75**–76
grids, **72**–74
smooth transitions, 70–71
themes, 69
Uniform Resource Locators. *See* URLs (Uniform Resource Locators)
unique visitors, 118
universal selector, **163,** 185
University of California, San Diego, 90
unordered lists, 247, 425
unvisited link colors, 370
uploading files with FTP (File Transfer Protocol), 138–140
URLs (Uniform Resource Locators), 9, 52, **125,** 145
background image, 375
complete *versus* partial, 125
conventions, 122–125
usability testing, 141–142
usable navigation, 400–402
user agents, 97
user-controlled font size, 93
user feedback, 120
users
browsers, 117
designing for, 77–91
detecting error, 97
expectations, 77, 85–86
general tendencies, 87
interacting with Web page content, 23
interaction, 80–82
locating, 401–402
perspective, 112
platforms, 117
profiling, 77–78
returning to site, 117
submitting file, 493–494
unfamiliar with site, 117
viewing Web pages, 82–87
user testing, 110

V

validators, **18,** 32, 36
valid code, **32,** 36
value attribute, 488, 491, 513
values, **153,** 172, 185
vector graphics, **343,** 385
vertical-align property, 215–217, 354
vertical margins, 254
vertical navigation bar, 429–431
vertical repeat, 377–378
video, 28
<video> elements, 24, 28
virtual gallery, 114
visited link colors, 370
visited links, 432–433
:visited pseudo-class, 370
visited state, 433
visual formatting model, 245–248
visual structure, 72–74
vspace attribute, 347–348

W

warm colors, 361
warning class, 209
Washington Post Web site, 65
WCAG (Web Content Accessibility Guidelines) recommendation, 94–97
W3 Counter Web site, 118
W3C Web site, 94
W3C (World Wide Web Consortium), **9**–10, 36
deprecated elements, 11
link validator, 141
polyglot document, 17
proposal for HTML5, 14
recommendations, **152**
Semantic Web Activity group, 31
standards, 51
SVG page, 342
validation service, 32
WCAG (Web Content Accessibility Guidelines) recommendation, 94–97
Web Accessibility Initiative, 94
XHMTL 1.0 (Extensible Hypertext Markup Language), **12**–13
XHTML 2.0 recommendation, 13
XML (Extensible Markup Language), 11–12
Web
broadband access, 51–52
dial-up users, 52
growth of, 10
need for standards, 9–11
universal medium, 31
Web Accessibility Initiative, 94
Web analytics, **117**–119, 145
Web browsers. *See* browsers, **5**
Web Content Accessibility Guidelines (WCAG). *See* WCAG (Web Content Accessibility Guidelines)
Web Decout Web site, 152
Web design environment, 49
browser cache and download time, 52
browser compatibility issues, 50–51
connection speed differences, 51–52
device and operating system issues, 53–54
Web designers, 49
Web fonts, 193–195
Web hosting package, 135
Web hosting services, 135–138
Web Hypertext Application Technology Working Group. *See* WHATWG (Web Hypertext Application Technology Working Group)
Webopedia, 400
Web Open Font Format. *See* WOFF (Web Open Font Format)
Web pages, **2,** 36
accessibility, 92–97
appearing and operating predictably, 97
background color, 372–373
backgrounds, 375–376
banner division, 297
centering in browser window, 297–298
common characteristics, 25–26
common sections, 2, 3
controlling length, 402
cross-platform compatibility, 6

divisions, 170–171
footer content, 27
header content, 27
HTM creation, 2–7
id or class names, 25
images, 27
interacting with content, 23
layout sections, 169–170
links, 8
loading external content, 22
logical divisions, 170–171
main division, 297
margins, 74
mockups, 108–109
multiple delivery platforms and presentation devices, 28
multiple divisions, 297
navigation division, 297
navigation elements, 27, 70
organizing information, 8
primary page content, 27
reflecting identity of site, 70
scalable, 199
sections, 27, 296–298
sizing graphics for, 352–353
standardizing display information, 151
standardizing naming conventions, 27–28
structure, 3–4
styles, 5–7
template creation, 108–109
testing in browsers, 5
users viewing, 82–87
valid code, **32**
visual structure, 72–74
white space, 74
wireframes, **109**
wrapper division, 297
zeroing margins, 255
Web palette, **359,** 385
Web servers, 52, **135,** 145
cookies, **29**
directory structure, 140
server logs, 117
Websiteoptimization.com, 52
Web sites
assessing effectiveness, 112
backups, 138
billboard, 112
blogs, 113
borrowing images, 345
budget, 112
catalog, 114
clients, 111
coded for browsers, 10
color schemes, 359–364
controlling page length, 402
CSS, 13
default main page name, 124
development process, 106–111
development team, 121–122
e-commerce, 114
flat hierarchy, 87
focus, 111
geographical results, 118
goals, 111
high-contrast version, 93
interoperability, 10
legacy style coding conventions, 13
limiting information overload, 402
limiting technical factors affecting, 112
look and feel, **66**–68
manageable information segments, 402
milestone dates, 112
navigation and content, 85–86
navigation pages, 87
new, 112
nonprofit organization, 113
objectives, 106
online shopping, 114
page views, **118**
portals, 113
product support, 115
promotion, 111
public interest, 113
publishing, 111, 113, 135
requirements, 112
reusing graphics, 53
RSS (Real Simple Syndication), 114
site map, 87
skills for building, 117
social networking, 113
software requirements, 120–121
special interest, 113
target audience, 112
testing, 53, 58, 140–142
three clicks rule, 87
transferring Web server, 135–140
unified design, 69–76
unique visitors, 118
upgrade, 112
virtual gallery, 114
wikis, 114
Web standards, 9
Web Standards project, 31
Web statistics, 120
Web Storage, 29
Web structure, 131
Web Style Guide, Third Edition, 85
Web Style Guide Web site, 193
Web workers, 28
well-formed document, **20,** 36
WestCIV Web site, 152
WHATWG (Web Hypertext Application Technology Working Group), 13
white space, 74
active, 75–76
block elements, 250
fieldsets, 518
images, 348, 356–357
between letters, 218
margins, 518
paragraphs, 222
passive, 75–76
tables, 463–464
text, 197, 215
white-space property, **222**
wide-screen displays, 55
width attribute, 346, 349–352
width property, 269–271, 274, 427, 455–456
Wikipedia
contextual linking, 419
hypertext links, 89
main page, 59
public domain stock photo Web sites, 344
wikis, 114
Windows, 29
Windows Web servers, 483
wireframes, **109,** 145
WOFF (Web Open Font Format), 195
wonderform.html file, 507–519
WordPress, 113
words
broken to wrap sentence, 223
spacing, 218–219
word-spacing property, 218–219
word-wrap property, **223**
World Wide Web Consortium. *See* W3C (World Wide Web Consortium)
wrapper, **297,** 331
wrapper division, 297
wrapper element, 310, 316, 318
wrapper style rule, 321
wrapping text, 222
writers, 122
WYSIWYG (What You *See* Is What You Get) HTML editors, 29

X

x-height, **196,** 238
XHTML 1.0, **12**–13
XHTML 2.0, 13
XHTML code, validating, 32
XHTML (Extensible Hypertext Markup Language), **12**–13, 21, 35
XHTML MIME type, 16
XHTML Namespace qualifier, 16
x-large keyword, 206
XML code, 12
XML documents, 12
XML (Extensible Markup Language), 11–12
XML syntax, 17–18
x-small keyword, 206
xx-large keyword, 206
xx-small keyword, 206

Z

zeroing margins, 254–255