4TH EDITION

Learning Web Design

A BEGINNER'S GUIDE TO HTML, CSS, JAVASCRIPT, AND WEB GRAPHICS

Jennifer Niederst Robbins

Whether you're a beginner or bringing your skills up to date, this book gives you a solid footing in modern web production. I teach each topic visually at a pleasant pace, with frequent exercises to let you try out new skills. Reading it feels like sitting in my classroom! —Jennifer Robbins

O'REILLY®

Obras protegidas por Direitos de Autor

Learning Web Design

Fourth Edition

A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics

Jennifer Niederst Robbins

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Obras protegidas por Direitos de Autor

Learning Web Design, Fourth Edition

by Jennifer Niederst Robbins

Copyright © 2012 Littlechair, Inc. All rights reserved.
Printed in Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Simon St. Laurent

Production Editor: Melanie Yarbrough

Copy Editor: Genevieve d'Entremont

Technical Reviewers: Aaron Gustafson, Matt Menzer, Joel Marsh

Interior Designer: Ron Bilodeau

Cover Designer: Mark Paglietti

Indexer: Ellen Troutman Zaig

Print History:

February 2001:	First edition.
March 2004:	Second edition.
June 2007:	Third edition.
August 2012:	Fourth edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. This book's trade dress is a trademark of O'Reilly Media, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-31927-4
[TI]
[2012-11-30]

Obras protegidas por Direitos de Autor

CONTENTS

Preface xi

Part I Getting Started

Chapter 1
Where Do I Start? 3
- Where Do I Start? 4
- What Does a Web Designer Do? 4
- What Languages Do I Need to Learn? 11
- What Do I Need to Buy? 14
- What You've Learned 19
- Test Yourself 20

Chapter 2
How the Web Works 21
- The Internet Versus the Web 21
- Serving Up Your Information 21
- A Word About Browsers 23
- Web Page Addresses (URLs) 24
- The Anatomy of a Web Page 26
- Putting It All Together 30
- Test Yourself 32

Chapter 3
Some Big Concepts You Need to Know 33
- A Dizzying Multitude of Devices 34
- Sticking with the Standards 36
- Progressive Enhancement 36
- Responsive Web Design 38

Obras protegidas por Direitos de Autor

One Web for All (Accessibility) 41
The Need for Speed (Site Performance) 43
Test Yourself 45

Part II HTML Markup for Structure

Chapter 4
Creating a Simple Page 49
A Web Page, Step by Step 49
Before We Begin, Launch a Text Editor 50
Step 1: Start with Content 53
Step 2: Give the Document Structure 55
Step 3: Identify Text Elements 58
Step 4: Add an Image 61
Step 5: Change the Look with a Style Sheet 64
When Good Pages Go Bad 65
Validating Your Documents 66
Test Yourself 67
Element Review: Document Structure 68

Chapter 5
Marking Up Text 69
Paragraphs 70
Headings 70
Lists 73
More Content Elements 76
Organizing Page Content 79
The Inline Element Roundup 84
Generic Elements (div and span) 95
Some Special Characters 99
Putting It All Together 100
Test Yourself 102
Element Review: Text 104

Chapter 6
Adding Links 105
The href Attribute 106
Linking to Pages on the Web 107
Linking Within Your Own Site 108
Targeting a New Browser Window 118
Mail Links 119
Telephone Links 120

Obras protegidas por Direitos de Autor

Test Yourself ... 121
Element Review: Links ... 122

Chapter 7
Adding Images ... 123
First, a Word on Image Formats ... 123
The img Element ... 124
A Window in a Window ... 130
Test Yourself ... 131
Element Review: Images ... 132

Chapter 8
Table Markup ... 133
How Tables Are Used ... 133
Minimal Table Structure ... 135
Spanning Cells ... 139
Table Accessibility ... 142
Wrapping Up Tables ... 144
Test Yourself ... 146
Element Review: Tables ... 146

Chapter 9
Forms ... 147
How Forms Work ... 147
The form Element ... 149
Variables and Content ... 151
The Great Form Control Roundup ... 152
Form Accessibility Features ... 171
Form Layout and Design ... 173
Test Yourself ... 175
Element Review: Forms ... 176

Chapter 10
What's Up, HTML5? ... 181
A Funny Thing Happened on the Way to XHTML 2 ... 182
In the Markup Department ... 185
Meet the APIs ... 189
Video and Audio ... 192
Canvas ... 198
Final Word ... 202
Test Yourself ... 203

Obras protegidas por Direitos de Autor

Part III CSS for Presentation

Chapter 11
Cascading Style Sheets Orientation 207

The Benefits of CSS 207
How Style Sheets Work 209
The Big Concepts 214
Moving Forward with CSS 221
Test Yourself 223

Chapter 12
Formatting Text 225

The Font Properties 225
Changing Text Color 243
A Few More Selector Types 244
Text Line Adjustments 249
Underlines and Other "Decorations" 252
Changing Capitalization 252
Spaced Out 253
Text Shadow 254
Changing List Bullets and Numbers 259
Test Yourself 261
CSS Review: Font and Text Properties 263

Chapter 13
Colors and Backgrounds 265

Specifying Color Values 265
Foreground Color 272
Background Color 273
Playing with Opacity 275
Introducing…Pseudo-Class Selectors 276
Pseudo-Element Selectors 279
Attribute Selectors 281
Background Images 284
The Shorthand background Property 293
Like a Rainbow (Gradients) 296
Finally, External Style Sheets 300
Test Yourself 303
CSS Review: Color and Background Properties 304

Obras protegidas por Direitos de Autor

Chapter 14
Thinking Inside the Box 305
The Element Box 305
Specifying Box Dimensions 306
Padding 312
Borders 316
Margins 328
Assigning Display Roles 333
Adding Drop Shadows to Boxes 335
Test Yourself 336
CSS Review: Basic Box Properties 338

Chapter 15
Floating and Positioning 341
Normal Flow 341
Floating 342
Positioning Basics 356
Relative Positioning 358
Absolute Positioning 359
Fixed Positioning 368
Test Yourself 370
CSS Review: Floating and Positioning Properties 371

Chapter 16
Page Layout with CSS 373
Page Layout Strategies 373
Page Layout Techniques 380
Multicolumn Layouts Using Floats 381
Positioned Layout 392
Top-to-Bottom Column Backgrounds 395
Test Yourself 398

Chapter 17
Transitions, Transforms, and Animation 399
Ease-y Does It (CSS Transitions) 399
CSS Transforms 410
Keyframe Animation 420
Test Yourself 423
CSS Review: Transitions, Transforms, and Animation 426

Obras protegidas por Direitos de Autor

Chapter 18
CSS Techniques 427
A Clean Slate (CSS Reset) 427
Image Replacement Techniques 429
CSS Sprites 430
Styling Forms 434
Styling Tables 441
Basic Responsive Web Design 444
Wrapping Up Style Sheets 454
Test Yourself 454
CSS Review: Table Properties 456

Part IV JavaScript for Behaviors

Chapter 19
Introduction to JavaScript 459
What Is JavaScript? 459
Adding JavaScript to a Page 463
The Anatomy of a Script 463
The Browser Object 478
Events 478
Putting It All Together 481
Test Yourself 483

Chapter 20
Using JavaScript 485
Meet the DOM 485
Polyfills 493
JavaScript Libraries 497
Big Finish 501
Test Yourself 502

Obras protegidas por Direitos de Autor

Part V Creating Web Graphics

Chapter 21
Web Graphics Basics 507
Image Sources 507
Meet the Formats 510
Image Size and Resolution 522
Working with Transparency 526
Introduction to SVG 533
Summing Up Images 538
Test Yourself 539

Chapter 22
Lean and Mean Web Graphics 541
General Image Optimization Strategies 542
Optimizing GIFs 543
Optimizing JPEGs 547
Optimizing PNGs 552
Optimize to File Size 553
Optimization in Review 554
Test Yourself 555

Appendix A
Answers 557

Appendix B
CSS3 Selectors 583

Index 587

Obras protegidas por Direitos de Autor

image not available

image not available

Marquis, who is a designer and developer at Filament Group, a member of the jQuery Mobile team, and the Technical Editor at *A List Apart*.

As in the first three editions, this book addresses the specific needs and concerns of beginners of all backgrounds, including seasoned graphic designers, programmers looking for a more creative outlet, office assistants, recent college graduates, work-at-home moms, and anyone else wanting to learn how to design websites. I've done my best to put the experience of sitting in my beginner web design class into a book, with exercises and tests along the way, so you get hands-on experience and can check your progress.

Whether you are reading this book on your own or using it as a companion to a web design course, I hope it gives you a good head start and that you have fun in the process.

How This Book Is Organized

Learning Web Design, Fourth Edition, is divided into five parts, each dealing with an important aspect of web development.

Part I: Getting Started

Part I lays a foundation for everything that follows in the book. I start off with some important general information about the web design environment, including the various roles you might play, the technologies you might learn, and tools that are available to you. You'll get your feet wet right away with HTML and CSS and learn how the Web and web pages generally work. I'll also introduce you to some Big Concepts that get you thinking the way modern web designers think about their craft.

Part II: HTML for Structure

The chapters in Part II cover the nitty-gritty of every element and attribute available to give content semantic structure, including the new elements introduced in HTML5. We'll cover the markup for text, links, images, tables, and forms. Part II closes out with an in-depth discussion of HTML5 and how it differs from previous standards.

Part III: CSS for Presentation

In the course of Part III, you'll go from learning the basics of using Cascading Style Sheets for changing the presentation of text to creating multicolumn layouts and even adding time-based animation and interactivity to the page. It also addresses common CSS techniques, including how to create a page using Responsive Web Design.

Part IV: JavaScript for Behaviors

Mat Marquis starts Part IV out with a rundown of JavaScript syntax so you can tell a variable from a function. You'll also get to know some ways that JavaScript is used, including DOM Scripting, and existing

Typographical Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Used to indicate URLs, email addresses, filenames, and directory names, as well as for emphasis.

Colored roman text

Used for special terms that are being defined and for cross-references.

`Constant width`

Used to indicate code examples and keyboard commands.

`Colored constant width`

Used for emphasis in code examples.

`Constant width italic`

Used to indicate placeholders for attribute and style sheet property values.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Where Do I Start?

Your particular starting point will no doubt depend on your background and goals. However, a good first step for everyone is to get a basic understanding of how the Web and web pages work. This book will give you that foundation. Once you learn the fundamentals, there are plenty of resources on the Web and in bookstores for you to further your learning in specific areas.

There are many levels of involvement in web design, from building a small site for yourself to making it a full-blown career. You may enjoy being a full-service website developer or just specializing in one skill. There are a lot of ways you can go.

If your involvement in web design is purely at the hobbyist level, or if you have just one or two web projects you'd like to publish, you may find that a combination of personal research (like reading this book), taking advantage of available templates, and perhaps even investing in a visual web design tool such as Adobe Dreamweaver may be all you need to accomplish the task at hand. Many Continuing Education programs offer introductory courses to web design and production.

If you are interested in pursuing web design or production as a career, you'll need to bring your skills up to a professional level. Employers may not require a web design degree, but they will expect to see working sample sites that demonstrate your skills and experience. These sites can be the result of class assignments, personal projects, or a simple site for a small business or organization. What's important is that they look professional and have well-written, clean HTML, style sheets, and possibly scripts behind the scenes. Getting an entry-level job and working as part of a team is a great way to learn how larger sites are constructed and can help you decide which aspects of web design you would like to pursue.

I Just Want a Blog!

You don't necessarily need to become a web designer to start publishing your words and pictures on the Web. You can start your own "blog" or personal journal site using one of the free or inexpensive blog hosting services. These services provide templates that generally spare you the need to learn HTML (although it still doesn't hurt). These are some of the most popular as of this writing:

- WordPress (*www.wordpress.com*)
- Blogger (*www.blogger.com*)
- Tumblr (*www.tumblr.com*)

Another drag-n-drop site design and hosting service that goes beyond the blog is Squarespace (*www.squarespace.com*).

What Does a Web Designer Do?

AT A GLANCE

The term "web design" has come to encompass a number of disciplines, including:

- Visual (graphic) design
- User interface and experience design
- Web document and style sheet production
- Scripting and programming
- Content strategy
- Multimedia

Over the years, the term "web design" has become a catchall for a process that encompasses a number of different disciplines, from user experience design, to document markup, to serious programming. This section describes some of the most common roles.

If you are designing a small website on your own, you will need to wear many hats. The good news is that you probably won't notice. Consider that the day-to-day upkeep of your household requires you to be part-time chef, housecleaner, accountant, diplomat, gardener, and construction worker—but to you it's just the stuff you do around the house. In the same way, as a solo web designer, you may be a part-time graphic designer, writer, HTML author, and information architect, but to you, it'll just feel like "making web pages." Nothing to worry about.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

If you are already a graphic designer, you will be able to adapt your skills to the Web easily, although this will not excuse you from acquiring a solid understanding of HTML, CSS, and other web technologies. Because most sites have at least a few images, even hobbyist web designers will need to know how to create and edit images, at minimum.

Again, I want to note that all of these responsibilities may fall into the hands of one designer who creates both the look and the functionality of a site. But for larger sites with bigger budgets, there is an opportunity to find your own special niche in the design process.

Development

A fair amount of the web design process involves the creation and trouble-shooting of the documents, style sheets, scripts, and images that make up a site. At web design firms, the team that handles the creation of the files that make up the website (or templates for pages that get assembled dynamically) is usually called the development or production department.

Web developers may not design the look or structure of the site themselves, but they do need to communicate well with designers and understand the intended site goals so they may suggest solutions that meet those goals.

The broad disciplines that fall under development are authoring, styling, and scripting/programming.

Authoring/markup

Authoring is the term used for the process of preparing content for delivery on the Web, or more specifically, marking up the content with HTML tags that describe its content and function. If you want a job as a web developer, you need to have an *intricate* knowledge of HTML and how it functions on various browsers and devices. The HTML specification is constantly evolving, which means you'll need to keep up with the latest best practices and opportunities as well as bugs and limitations. The good news is, it's not difficult to get started, and from there, you can gradually increase your skills. We'll be dabbling with HTML in Chapter 2, How the Web Works, and then discussing it in great detail in Part II of this book.

Styling

In web design, the appearance of the page in the browser is controlled by style rules written in CSS (Cascading Style Sheets). We'll get deep into CSS in Part III of this book (including what "cascading" means!), but for now just know that in contemporary web design, the appearance of the page is handled separately from the HTML markup of the page. Again, if you are interested in working in web development, knowing your way around CSS and how it is supported (or not supported) by browsers is guaranteed to be part of your job description.

Style Tiles

Another approach to capturing the look and feel of a site is to create style tiles, which give examples of color schemes, branding elements, content and UI treatments, and mood boards without applying them to a specific page layout. The idea is to agree upon a consistent visual language for the site. For more on this technique, read the article "Style Tiles and How They Work," by Samantha Warren (*www.alistapart.com/articles/style-tiles-and-how-they-work/*), and visit her excellent site where you can download a template at *styletil.es*.

NOTE

Many visual designers translate their designs into HTML and CSS documents themselves. In fact, there is a popular argument that in order to call yourself a "web designer," you must be able to build your designs yourself, and nearly everyone agrees that your job prospects will be better if you are able to code as well as design.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Hypertext Markup Language (HTML)

You may see HTML and XHTML referred to collectively as (X)HTML.

HTML (HyperText Markup Language) is the language used to create web page documents. There are a few versions of HTML in use today: HTML 4.01 is the most firmly established, and the newer, more robust HTML5 is gaining steam and browser support. Both versions have a stricter implementation called XHTML (eXtensible HTML), which is essentially the same language with much stricter syntax rules. We'll get to the particulars of what makes the various versions different in Chapter 10, What's Up, HTML5?.

HTML is not a programming language; it is a markup language, which means it is a system for identifying and describing the various components of a document such as headings, paragraphs, and lists. The markup indicates the document's underlying structure (you can think of it as a detailed, machine-readable outline). You don't need programming skills—only patience and common sense—to write HTML.

The best way to learn HTML is to write out some pages by hand, as we will be doing in the exercises in this book. If you end up working in web production, you'll live and breathe HTML. But even hobbyists will benefit from knowing what is going on under the hood. The good news is that it's simple to learn the basics.

Cascading Style Sheets (CSS)

While HTML is used to describe the content in a web page, it is Cascading Style Sheets (CSS) that describe how that content should *look*. In the web design biz, the way the page looks is known as its presentation. That means fonts, colors, background images, line spacing, page layout, and so on, are all controlled with CSS. With the newest version (CSS3), you can even add special effects and basic animation to your page.

CSS also provides methods for controlling how documents will be presented in contexts other than the traditional desktop browser, such as in print or on devices with small screen widths. It also has rules for specifying the non-visual presentation of documents, such as how they will sound when read by a screen reader (although those are not well supported).

Style sheets are also a great tool for automating production because you can change the way an element looks across all the pages in your site by editing a single style sheet document. Style sheets are supported to some degree by all modern browsers.

NOTE

When this book uses the term "style sheets," it is always referring to Cascading Style Sheets, the standard style sheet language for the World Wide Web.

Although it is possible to publish web pages using HTML alone, you'll probably want to take on style sheets so you're not stuck with the browser's default styles. If you're looking into designing websites professionally, proficiency at style sheets is mandatory.

Style sheets are discussed further in Part III.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Software

There's no shortage of software available for creating web pages. In the early days, we just made do with tools originally designed for print. Today, there are wonderful tools created specifically with web design in mind that make the process more efficient. Although I can't list every available software release, I'd like to introduce you to the most common and proven web design tools. Note that you can download trial versions of many of these programs from the company websites, as listed in the "Popular Web Design Software Links" sidebar later in this chapter.

Web page authoring

Web-authoring tools are similar to desktop publishing tools, but the end product is a web page (an HTML file and its supporting files). These tools provide a visual "WYSIWYG" (What You See Is What You Get, pronounced "whizzy-wig") interface and shortcuts that save you from typing repetitive HTML and CSS. These tools won't excuse you from learning HTML. Even the most sophisticated tools won't generate HTML as clean or well considered as a professional writing by hand, but they can speed up the process once you know what you're doing.

NOTE

To do the exercises in this book, all you'll need is the text editor that came with your operating system. No special programs are required.

The following are some popular web-authoring programs:

Adobe Dreamweaver. This is the hands-down industry standard due to its relatively clean code and advanced features.

Microsoft Expression Web (Windows only). Part of Microsoft's suite of professional design tools, MS Expression Web boasts standards-compliant code and CSS-based layouts.

Nvu (Linux, Windows, and Mac OS X). Don't want to pay for a WYSIWYG editor? Nvu (pronounced N-view, for "new view") is an open source tool that matches many of the features in Dreamweaver, and you can download it for free at *nvu.com*.

HTML editors

HTML editors (as opposed to WYSIWYG authoring tools) are designed to speed up the process of writing HTML by hand. They do not allow you to edit the page visually, so you need to check your work in a browser. Many professional web designers actually prefer to author HTML documents by hand, and they tend to recommend the following:

TextPad (Windows only). TextPad is a simple and inexpensive plain-text code editor for Windows.

Sublime Text (Window, Mac, Linux). This inexpensive and up-and-coming text editor looks stripped down but has a lot of functionality (like color coding and full code overviews) that developers love.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

exercise 1-1 | Taking stock

Now that you're taking that first step in learning web design, it might be a good time to take stock of your assets and goals. Using the lists in this chapter as a general guide, try jotting down answers to the following questions:

- What are your web design goals? To become a professional web designer? To make personal websites only?
- Which aspects of web design interest you the most?
- What current skills do you have that will be useful in creating web pages?
- Which skills will you need to brush up on?
- Which hardware and software tools do you already have for web design?
- Which tools do you need to buy? Which tools would you like to buy eventually?

Test Yourself

Each chapter in this book ends with a few questions that you can answer to see if you picked up the important bits of information. Answers appear in Appendix A.

1. Match these web professionals with the final product they might be responsible for producing.

A. Graphic designer	_____ HTML and CSS documents
B. Production department	_____ PHP scripts
C. User experience designer	_____ Photoshop page sketch
D. Web programmer	_____ Storyboards

2. What does the W3C do?

3. Match the web technology with its appropriate task:

A. HTML	_____ Checks a form field for a valid entry
B. CSS	_____ Creates a custom server-side web application
C. JavaScript	_____ Identifies text as a second-level heading
D. PHP	_____ Defines a new markup language for sharing financial information
E. XML	_____ Makes all second-level headings blue

4. What is the difference between frontend and backend web development?

5. What is the difference between a web-authoring program and an HTML-editing tool?

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Web Page Addresses (URLs)

Every page and resource on the Web has its own special address called a URL, which stands for Uniform Resource Locator. It's nearly impossible to get through a day without seeing a URL (pronounced "U-R-L," not "erl") plastered on the side of a bus, printed on a business card, or broadcast on a television commercial. Web addresses are fully integrated into modern vernacular.

Some URLs are short and sweet. Others may look like crazy strings of characters separated by dots (periods) and slashes, but each part has a specific purpose. Let's pick one apart.

The parts of a URL

A complete URL is generally made up of three components: the protocol, the site name, and the absolute path to the document or resource, as shown in Figure 2-1.

Figure 2-1. The parts of a URL.

❶ `http://`

The first thing the URL does is define the protocol that will be used for that particular transaction. The letters HTTP let the server know to use HyperText Transfer Protocol, or get into "web mode."

❷ `www.example.com`

The next portion of the URL identifies the website by its domain name. In this example, the domain name is *example.com*. The "www." part at the beginning is the particular host name at that domain. The host name "www" has become a convention, but is not a rule. In fact, sometimes the host name may be omitted. There can be more than one website at a domain (sometimes called subdomains). For example, there might also be *development.example.com*, *clients.example.com*, and so on.

❸ `/2012/samples/first.html`

This is the absolute path through directories on the server to the requested HTML document, *first.html*. The words separated by slashes are the directory names, starting with the root directory of the host (as indicated by the initial /). Because the Internet originally comprised computers running the Unix operating system, our current way of doing things still

Hey, There's No http:// on That URL!

Because nearly all web pages use the Hypertext Transfer Protocol, the `http://` part is often just implied. This is the case when site names are advertised in print or on TV, as a way to keep the URL easy to remember.

Additionally, browsers are programmed to add `http://` automatically as a convenience to save you some keystrokes. It may seem like you're leaving it out, but it is being sent to the server behind the scenes.

When we begin using URLs to create hyperlinks in HTML documents in Chapter 6, Adding Links, you'll learn that it is necessary to include the protocol when making a link to a web page on another server.

NOTE

Sometimes you'll see a URL that begins with `https://`*. This is an indication that it is a secure server transaction. Secure servers have special encryption devices that hide delicate content, such as credit card numbers, while they are transferred to and from the browser. Look for it the next time you're shopping online.*

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

image not available

image not available

image not available

image not available

image not available

image not available

image not available

image not available

- The Future Friendly site (*futurefriend.ly*) includes a call to arms composed by many of the brightest mobile designers of the day. They concluded that with the landscape changing so rapidly, we can't make our designs future-proof, but we can make them "future friendly." They assemble a number of tips and resources for doing so.

Sticking with the Standards

Sticking with web standards is your primary tool for ensuring your site is as consistent as possible.

So how do we deal with this diversity? One good start is to follow the HTML, CSS, and JavaScript standards as documented by the World Wide Web Consortium (W3C). Sticking with web standards is your primary tool for ensuring your site is as consistent as possible on all standards-compliant browsers (that's approximately 99% of browsers in current use). It also helps make your content forward-compatible as web technologies and browser capabilities evolve. Another benefit is you can tell your clients that you create "standards-compliant" sites, and they will like you more.

The notion of standards compliance may seem like a no-brainer, but it used to be that everyone, including the browser makers, played fast and loose with HTML and scripting. The price we paid was incompatible browser implementations and the need to create sites twice to make them work for everyone. I talk more about web standards throughout this book, so I won't go into too much detail here. Suffice it to say that the web standards are your friends. Everything you learn in this book will get you headed in the right direction.

For further reading

The bible for standards compliance and how it makes good business sense is *Designing with Web Standards* by Jeffrey Zeldman (New Riders). Go read it (when you're done with this book, of course).

Progressive Enhancement

Progressive enhancement is a strategy for coping with unknown browser capabilities.

With a multitude of browsers comes a multitude of levels of support for the web standards. In fact, no browser has implemented all the standards 100%, and there are always new technologies that are slowly gaining steam. Furthermore, users can set their own browser preferences, so they may have a browser that supports JavaScript but have chosen to turn it off. The point here is that we are faced with a wide range of browser capabilities—from basic HTML support only to all the bells and whistles.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

image not available

image not available

image not available

image not available

widgets, Facebook Like buttons, and advertisements can make dozens of server requests each. You may be surprised to see how many server requests even a simple site makes.

If you'd like to see for yourself, you can use the developer tool in the Chrome browser to see each request to the server and how many milliseconds it takes. Here's how you do it:

1. Launch the Chrome browser and go to any web page.
2. Go to the View menu and select Developer → Developer Tools. A panel will open at the bottom of the browser.
3. Select the Network tab in the tools view and reload the page. The chart (commonly referred to as a waterfall chart) shows you all the requests made and assets downloaded. The columns on the right show the amount of time each request took in milliseconds. At the bottom of the chart, you can see a summary of the number of requests made and the total amount of data transferred.

Figure 3-5 shows a portion of the performance waterfall chart for my site, *Jenville.com*, which is a simple site (but not as simple as I thought!). You can poke around any site on the Web this way. It can be very educational.

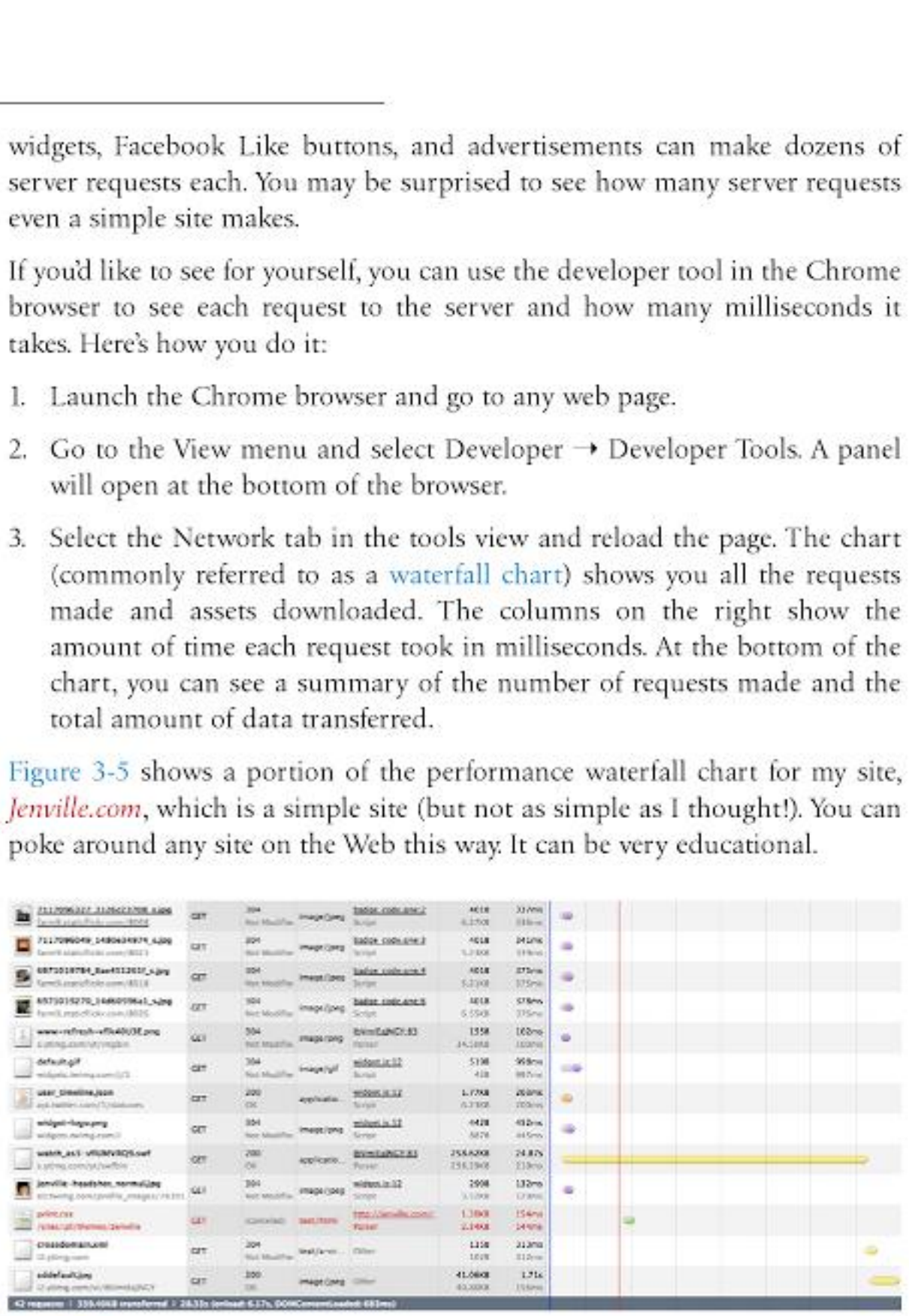


Figure 3-5. Waterfall charts such as this one created by the Chrome Network developer tool show you the individual server requests made by a web page and the amount of time each request takes.

I won't address site performance much in this book, but I do want you to remember the importance of keeping file sizes as small as possible and eliminating unnecessary server requests in your web design work.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Creating a new document in TextEdit (Mac OS X)

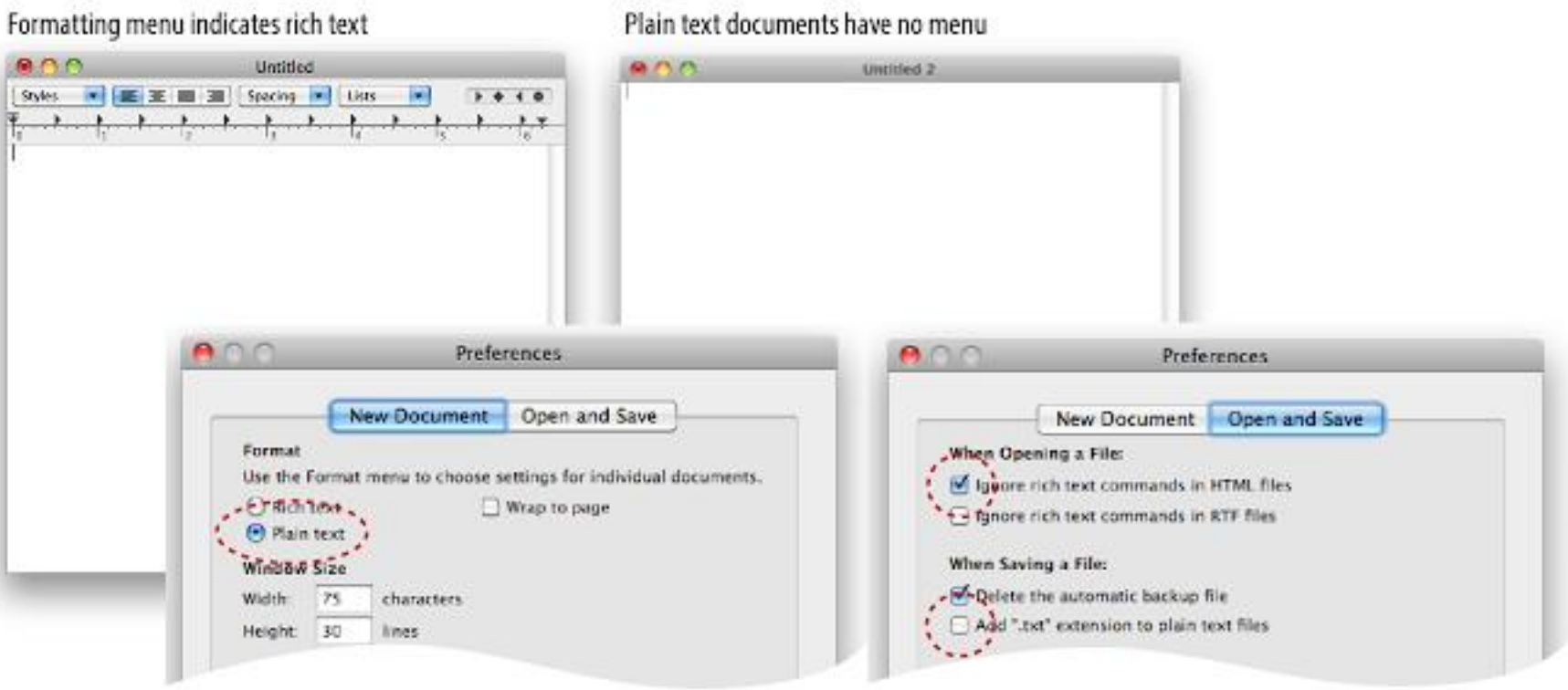
By default, TextEdit creates "rich text" documents—that is, documents that have hidden style formatting instructions for making text bold, setting font size, and so on. You can tell that TextEdit is in rich-text mode when it has a formatting toolbar at the top of the window (plain-text mode does not). HTML documents need to be plain-text documents, so we'll need to change the format, as shown in this example (Figure 4-3).

1. Use the Finder to look in the *Applications* folder for TextEdit. When you've found it, double-click the name or icon to launch the application.
2. TextEdit opens a new document. The text-formatting menu at the top shows that you are in Rich Text mode. Here's how you change it.
3. Open the Preferences dialog box from the TextEdit menu.
4. There are three settings you need to adjust:

   On the "New Document" tab, select "Plain text".

   On the "Open and Save" tab, select "Ignore rich text commands in HTML files" and turn off "Append '.txt' extensions to plain text files".
5. When you are done, click the red button in the top-left corner.
6. When you create a new document, the formatting menu will no longer be there and you can save your text as an HTML document. You can always convert a document back to rich text by selecting Format → Make Rich Text when you are not using TextEdit for HTML.

Figure 4-3. Launching TextEdit and choosing Plain Text settings in the Preferences.



Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Basic document structure

Figure 4-7 shows the recommended minimal skeleton of an HTML5 document. I say "recommended" because the only element that is *required* in HTML is the **title**. But I feel it is better, particularly for beginners, to explicitly organize documents with the proper structural markup. And if you are writing in the stricter XHTML, all of the following elements except **meta** must be included in order to be valid. Let's take a look at what's going on in Figure 4-7.

Figure 4-7. The minimal structure of an HTML document.

❶ I don't want to confuse things, but the first line in the example isn't an element at all; it is a document type declaration (also called DOCTYPE declaration) that identifies this document as an HTML5 document. I have a lot more to say about DOCTYPE declarations in Chapter 10, What's Up, HTML5?, but for this discussion, suffice it to say that including it lets modern browsers know they should interpret the document as written according to the HTML5 specification.

❷ The entire document is contained within an **html** element. The **html** element is called the root element because it contains all the elements in the document, and it may not be contained within any other element. It is used for both HTML and XHTML documents.

❸ Within the **html** element, the document is divided into a head and a body. The **head** element contains descriptive information about the document itself, such as its title, the style sheet(s) it uses, scripts, and other types of "meta" information.

NOTE

Prior to HTML5, the syntax for specifying the character set with the **meta** *element was a bit more elaborate. If you are writing your documents in HTML 4.01 or XHTML 1.0, your* **meta** *element should look like this:*

```
<meta http-equiv="content-
type" content="text/html;
charset=UTF-8">
```

❹ The **meta** elements within the **head** element provide information *about* the document itself. A **meta** element can be used to provide all sorts of information, but in this case, it specifies the character encoding (the standardized collection of letters, numbers, and symbols) used in the document. I don't want to go into too much detail on this right now, but know that there are many good reasons for specifying the **charset** in every document, so I have included it as part of the minimal document structure.

❺ Also in the **head** is the mandatory **title** element. According to the HTML specification, every document must contain a descriptive title.

❻ Finally, the **body** element contains everything that we want to show up in the browser window.

Are you ready to add some structure to the Black Goose Bistro home page? Open the *index.html* document and move on to Exercise 4-2.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

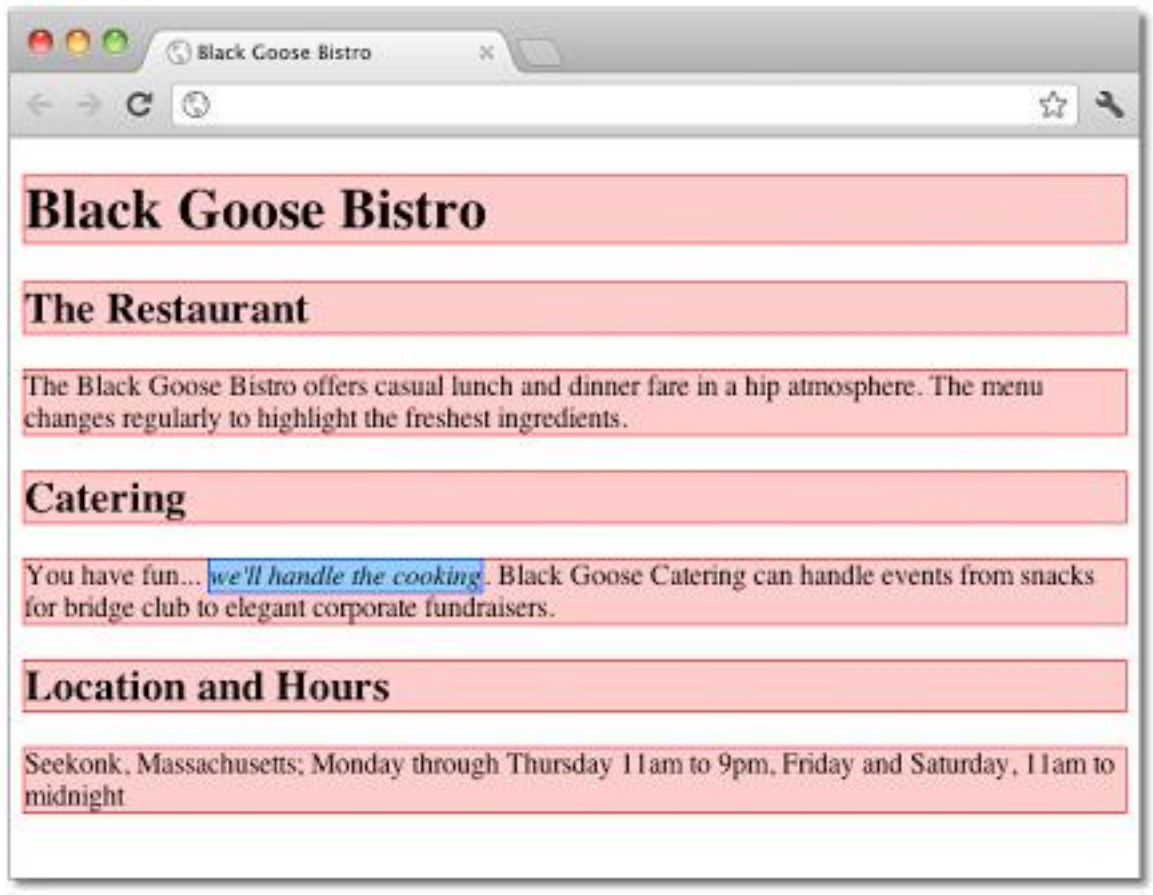


Figure 4-10. The outlines show the structure of the elements in the home page.

By contrast, look at the text we marked up as emphasized (`em`). It does not start a new line, but rather stays in the flow of the paragraph. That is because the `em` element is an inline element. Inline elements do not start new lines; they just go with the flow. In Figure 4-10, the inline `em` element is outlined in light blue.

Default styles

The other thing that you will notice about the marked-up page in Figures 4-9 and 4-10 is that the browser makes an attempt to give the page some visual hierarchy by making the first-level heading the biggest and boldest thing on the page, with the second-level headings slightly smaller, and so on.

How does the browser determine what an `h1` should look like? It uses a style sheet! All browsers have their own built-in style sheets (called user agent style sheets in the spec) that describe the default rendering of elements. The default rendering is similar from browser to browser (for example, `h1`s are always big and bold), but there are some variations (long quotes may or may not be indented).

If you think the `h1` is too big and clunky as the browser renders it, just change it with a style sheet rule. Resist the urge to mark up the heading with another element just to get it to look better, for example, using an `h3` instead of an `h1` so it isn't as large. In the days before ubiquitous style sheet support, elements were abused in just that way. Now that there are style sheets for controlling the design, you should always choose elements based on how

Adding Hidden Comments

You can leave notes in the source document for yourself and others by marking them up as comments. Anything you put between comment tags (`<!-- -->`) will not display in the browser and will not have any effect on the rest of the source.

```
<!-- This is a comment -->
<!-- This is a
  multiple-line comment
  that ends here. -->
```

Comments are useful for labeling and organizing long documents, particularly when they are shared by a team of developers. In this example, comments are used to point out the section of the source that contains the navigation.

```
<!-- start global nav -->
<ul>
  ...
</ul>
<!-- end global nav -->
```

Bear in mind that although the browser will not display comments in the web page, readers can see them if they "view source," so be sure that the comments you leave are appropriate for everyone. It's probably a good idea just to strip out notes to your fellow developers before the site is published. It cuts some bytes off the file size as well.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

3. I'd like the image to appear above the title, so let's add a line break (`br`) after the `img` element to start the headline text on a new line.

```
<h1><img src="blackgoose.png" alt="Black Goose logo"><br>Black
Goose Bistro</h1>
```

4. Let's break up the last paragraph into three lines for better clarity. Drop a `<br>` tag at the spots you'd like the line breaks to occur. Try to match the screenshot in Figure 4-14.
5. Now save *index.html* and open or refresh it in the browser window. The page should look like the one shown in Figure 4-14. If it doesn't, check to make sure that the image file, *blackgoose.png*, is in the same directory as *index.html*. If it is, then check to make sure that you aren't missing any characters, such as a closing quote or bracket, in the `img` element markup.

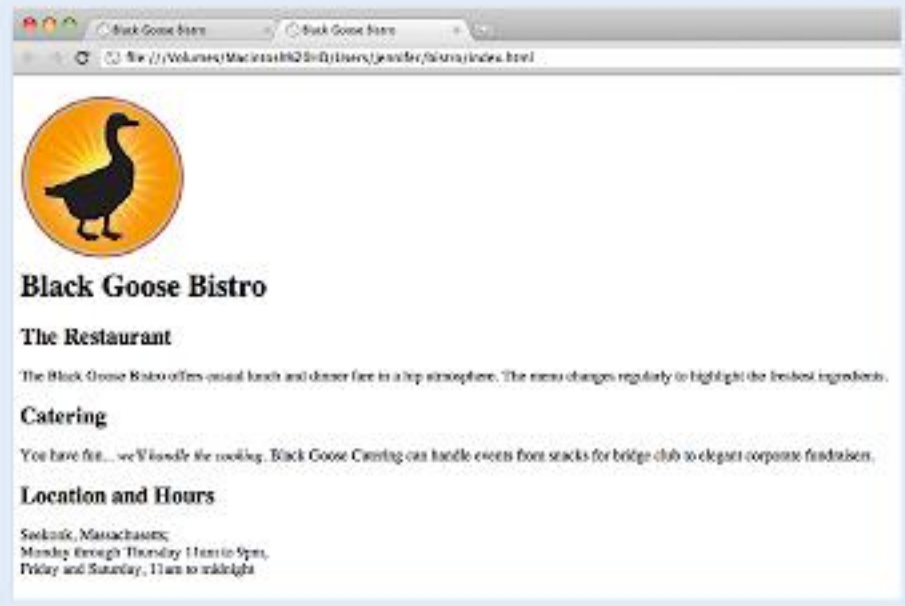


Figure 4-14. The Black Goose Bistro page with the logo image.

Step 5: Change the Look with a Style Sheet

Depending on the content and purpose of your website, you may decide that the browser's default rendering of your document is perfectly adequate. However, I think I'd like to pretty up the Black Goose Bistro home page a bit to make a good first impression on potential patrons. "Prettying up" is just my way of saying that I'd like to change its presentation, which is the job of Cascading Style Sheets (CSS).

In Exercise 4-5, we'll change the appearance of the text elements and the page background using some simple style sheet rules. Don't worry about understanding them all right now; we'll get into CSS in more detail in Part III. But I want to at least give you a taste of what it means to add a "layer" of presentation onto the structure we've created with our markup.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

3. Indicate whether each of these filenames is an acceptable name for a web document by circling "Yes" or "No." If it is not acceptable, provide the reason why.

a. *Sunflower.html*	Yes	No
b. *index.doc*	Yes	No
c. *cooking home page.html*	Yes	No
d. *Song_Lyrics.html*	Yes	No
e. *games/rubix.html*	Yes	No
f. *%whatever.html*	Yes	No

4. All of the following markup examples are incorrect. Describe what is wrong with each one, and then write it correctly.

   a. `<img "birthday.jpg">`

   b. `<i>Congratulations!<i>`

   c. `<a href="file.html">linked text</a href="file.html">`

   d. `<p>This is a new paragraph<\p>`

5. How would you mark up this comment in an HTML document so that it doesn't display in the browser window?

```
product list begins here
```

Element Review: Document Structure

This chapter introduced the elements that establish the structure of the document. The remaining elements introduced in the exercises will be treated in more depth in the following chapters.

Element	Description
`body`	Identifies the body of the document that holds the content
`head`	Identifies the head of the document that contains information about the document
`html`	The root element that contains all the other elements
`meta`	Provides information about the document
`title`	Gives the page a title

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

Indicating a Shift in Themes

`<hr>`

A horizontal rule

If you want to indicate that one topic or thought has completed and another one is beginning, you can insert what is called in HTML5 a "paragraph-level thematic break" using the **hr** element. The **hr** element adds a logical divider between sections of a page or paragraphs of text without introducing a new heading level.

In HTML versions prior to HTML5, **hr** was defined as a "horizontal rule" because it inserted a horizontal line on the page. Browsers still render **hr** as a 3D shaded rule and put it on a line by itself with some space above and below by default, but it now has a new semantic purpose. If a decorative line is all you're after, it is better to create a rule by specifying a colored border before or after an element with CSS.

hr is an empty element—you just drop it into place where you want the thematic break to occur, as shown in this example and Figure 5-2. Note that in XHTML, the **hr** element must be closed with a slash: **<hr />**.

```
<h3>Times</h3>
<p>Description and history of the Times typeface.</p>
<hr>
<h3>Georgia</h3>
<p>Description and history of the Georgia typeface.</p>
```

Times

Description and history of the Times typeface.

Georgia

Description and history of the Georgia typeface.

Figure 5-2. The default rendering of a thematic break (horizontal rule).

Heading groups

`<hgroup>...</hgroup>`

A group of stacked headings

NEW IN HTML5

It is common for headlines to have clarifying subheads or taglines. Take, for example, the title of Chapter 4 in this book:

Creating a Simple Page
(HTML Overview)

In the past, marking stacks of headings and subheadings was somewhat problematic. The first line, "Creating a Simple Page," is clearly an **h1**, but if you make the second line an **h2**, you may introduce an unintended new level to the document outline. The best you could do was mark it as a paragraph, but that didn't exactly make semantic sense.

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

```
    <p>Postscript emerged as a standard due to its support of
graphics and its early support on the Macintosh computer and Apple
laser printer.</p>
  </dd>
</dl>
```

Linotype
: Line-casting allowed type to be selected, used, then recirculated into the machine automatically. This advance increased the speed of typesetting and printing dramatically.

Photocomposition
: Typefaces are stored on film then projected onto photo-sensitive paper. Lenses adjust the size of the type.

Digital type
: Digital typefaces store the outline of the font shape in a format such as Postscript. The outline may may be scaled to any size for output.

  Postscript emerged as a standard due to its support of graphics and its early support on the Macintosh computer and Apple laser printer.

Figure 5-6. The default rendering of a definition list. Definitions are set off from the terms by an indent.

The `dl` element is only allowed to contain `dt` and `dd` elements. It is OK to have multiple definitions with one term and vice versa. You cannot put headings or content-grouping elements (like paragraphs) in names (`dt`), but the value (`dd`) can contain any type of flow content.

More Content Elements

We've covered paragraphs, headings, and lists, but there are a few more special text elements to add to your HTML toolbox that don't fit into a neat category: long quotations (`blockquote`), preformatted text (`pre`), and figures (`figure` and `figcaption`). One thing these elements do have in common is that they are considered "grouping content" in the HTML5 spec (along with `p`, `hr`, the list elements, and the generic `div`, covered later in this chapter). The other thing they share is that browsers typically display them as block elements by default.

Long quotations

`<blockquote>...</blockquote>`

A lengthy, block-level quotation

If you have a long quotation, a testimonial, or a section of copy from another source, you should mark it up as a `blockquote` element. It is recommended that content within `blockquote` elements be contained in other elements, such as paragraphs, headings, or lists, as shown in this example (see the sidebar "Sectioning Roots").

Sectioning Roots

The `blockquote` is in a category of elements called sectioning roots. Headings in a sectioning root element will not be included in the main document outline. That means you can have a complex heading hierarchy within a `blockquote` without worrying how it will affect the overall structure of the document. Other sectioning root elements include `figure`, `details`, `fieldset` (for organizing form fields), `td` (a table cell), and `body` (because it has its own outline, which also happens to be the outline of the document).

Obras protegidas por Direitos de Autor

image not available

image not available

image not available

image not available

image not available

image not available

image not available

it come last in the document or sectioning element. It could also appear at or near the beginning if it makes semantic sense.

NOTE

You can also add headers and footers to sectioning root elements: `body`, `blockquote`, `details`, `figure`, `td`, *and* `fieldset`.

In this simple example, we see the typical information listed at the bottom of an article or blog post marked up as a **`footer`**.

```
<article>
  <header>
    <h1>More about WOFF</h1>
    <p>by Jennifer Robbins, <time datetime="11-11-2011"
    pubdate>November 11, 2011</time></p>
  </header>
  <p>...article content starts here…</p>
  <footer>
    <p><small>Copyright &copy;2012 Jennifer Robbins.</small></p>
    <nav>
    <ul>
      <li><a href="">Previous</a></li>
      <li><a href="">Next</a></li>
    </ul>
    </nav>
  </footer>
</article>
```

Addresses

`<address>...</address>`

Contact information

Last, and well, least, is the **`address`** element that is used to create an area for contact information for the author or maintainer of the document. It is generally placed at the end of the document or in a section or article within a document. An **`address`** would be right at home in a **`footer`** element.

It is important to note that the **`address`** element should *not* be used for any old address on a page, such as mailing addresses. It is intended specifically for author contact information (although that could potentially be a mailing address). Following is an example of its intended use. The "a href" parts are the markup for links...we'll get to those in Chapter 6, Adding Links.

NOTE

You'll get a chance to try out the section elements in Exercise 5-3 at the end of this chapter.

```
<address>
Contributed by <a href="../authors/robbins/">Jennifer Robbins</a>,
<a href="http://www.oreilly.com/">O'Reilly Media</a>
</address>
```

The Inline Element Roundup

Now that we've identified the larger chunks of content, we can provide semantic meaning to phrases within the chunks using what HTML5 calls text-level semantic elements. On the street, you are likely to hear them called inline elements because they display in the flow of text by default and do not cause any line breaks. That's also how they were referred to in HTML versions prior to HTML5.

Obras protegidas por Direitos de Autor

Text-level (inline) elements

Despite all the types of information you could add to a document, there are only a couple dozen text-level semantic elements in HTML5. Table 5-1 lists all of them.

Table 5-1. Text-level semantic elements

Element	Description
a	An anchor or hypertext link (see Chapter 6 for details)
abbr	Abbreviation
b	Added visual attention, such as keywords (bold)
bdi	NEW IN HTML5 Indicates text that may have directional requirements
bdo	Bidirectional override; explicitly indicates text direction (left to right, ltr, or right to left, rtl)
br	Line break
cite	Citation; a reference to the title of a work, such as a book title
code	Computer code sample
data	WHATWG ONLY Machine-readable equivalent dates, time, weights, and other measurable values
del	Deleted text; indicates an edit made to a document
dfn	The defining instance or first occurrence of a term
em	Emphasized text
i	Alternative voice (italic)
ins	Inserted text; indicates an insertion in a document
kbd	Keyboard; text entered by a user (for technical documents)
mark	NEW IN HTML5 Contextually relevant text
q	Short, inline quotation
ruby, rt, rp	NEW IN HTML5 Provides annotations or pronunciation guides under East Asian typography and ideographs
s	Incorrect text (strike-through)
samp	Sample output from programs
small	Small print, such as a copyright or legal notice (displayed in a smaller type size)
span	Generic phrase content
strong	Content of strong importance
sub	Subscript
sup	Superscript
time	NEW IN HTML5 Machine-readable time data
u	Text that would normally be underlined, such as a formal name or misspelled word
var	A variable or program argument (for technical documents)
wbr	Word break

The Inline Elements Backstory

Many of the inline elements that have been around since the dawn of the Web were introduced to change the visual formatting of text selections due to the lack of a style sheet system. If you wanted bolded text, you marked it as **b**. Italics? Use the **i** element. In fact, there was once a **font** element used solely to change the font, color, and size of text (the horror!). Not surprisingly, HTML5 kicked the purely presentational **font** element to the curb. However, many of the old-school presentational inline elements (for example, **u** for underline and **s** for strike-through) have been kept in HTML5 and given new semantic definitions (**b** is now for "keywords," **s** for "inaccurate text").

Some inline elements are purely semantic (such as **abbr** or **time**) and don't have default renderings. For these, you'll need to use CSS rules if you want to change the way they display.

In the element descriptions in this section, I'll provide both the definition of the inline elements and the expected browser default rendering if there is one.

Obras protegidas por Direitos de Autor

Obsolete HTML 4.01 Text Elements

HTML5 finally retired many elements that were marked as deprecated (phased out and discouraged from use) in HTML 4.01. For the sake of thoroughness, I include them here in case you run across them in legacy markup or web authoring tool. But there's no reason to use them—most have analogous style sheet properties or are simply poorly supported.

Element	Description
`acronym`	Indicates an acronym (e.g., NASA); authors should use **`abbr`** instead
`applet`	Inserts a Java applet
`basefont`	Establishes default font settings for a document
`big`	Makes text slightly larger than default text size
`center`	Centers content horizontally
`dir`	Directory list (replaced by unordered lists)
`font`	Font face, color, and size
`isindex`	Inserts a search box
`menu`	Menu list (replaced by unordered lists; however, **`menu`** is now used to provide contextual menu commands)
`strike`	Strike-through text
`tt`	Teletype; displays in constant-width font

Emphasized text

`<em>...</em>`

Stressed emphasis

Use the **`em`** element to indicate which part of a sentence should be stressed or emphasized. The placement of **`em`** elements affects how a sentence's meaning is interpreted. Consider the following sentences that are identical, except for which words are stressed.

```
<p><em>Matt</em> is very smart.</p>
<p>Matt is <em>very</em> smart.</p>
```

The first sentence indicates *who* is very smart. The second example is about *how* smart he is.

Emphasized text (**`em`**) elements nearly always display in italics by default (Figure 5-9), but of course you can make them display any way you like with a style sheet. Screen readers may use a different tone of voice to convey stressed content, which is why you should use an **`em`** element only when it makes sense semantically, not just to achieve italic text.

Important text

`<strong>...</strong>`

Strong importance

The **`strong`** element indicates that a word or phrase is important. In the following example, the **`strong`** element identifies the portion of instructions that requires extra attention.

Obras protegidas por Direitos de Autor

```
<p>When checking out of the hotel, <strong>drop the keys in the red box
by the front desk</strong>.</p>
```

Visual browsers typically display **strong** text elements in bold text by default. Screen readers may use a distinct tone of voice for important content, so mark text as **strong** only when it makes sense semantically, not just to make text bold.

The following is a brief example of our **em** and **strong** text examples. Figure 5-9 should hold no surprises.

***Matt* is very smart.**

Matt is *very* smart.

When returning the car, drop the keys in the red box by the front desk.

Figure 5-9. The default rendering of emphasized and strong text.

The previously presentational elements that are sticking around in HTML5 with fancy new semantic definitions

As long as we're talking about bold and italic text, let's see what the old **b** and **i** elements are up to now. The elements **b**, **i**, **u**, **s**, and **small** were introduced in the old days of the Web as a way to provide typesetting instructions (bold, italic, underline, strike-through, and smaller text, respectively). Despite their original presentational purposes, these elements have been included in HTML5 and given updated, semantic definitions based on patterns of how they've been used. Browsers still render them by default as you'd expect (Figure 5-10). However, if a type style change is all you're after, using a style sheet rule is the appropriate solution. Save these for when they are semantically appropriate.

Let's look at these elements and their correct usage, as well as the style sheet alternatives.

`<b>...</b>`

Keywords or visually emphasized text (bold)

`<i>...</i>`

Alternative voice (italic)

`<s>...</s>`

Incorrect text (strike-through)

`<u>...</u>`

Annotated text (underline)

`<small>...</small>`

Legal text; small print (smaller type size)

`b`

HTML 4.01 definition: Bold

HTML5 definition: Keywords, product names, and other phrases that need to stand out from the surrounding text without conveying added importance or emphasis.

CSS alternative: For bold text, use **font-weight**. Example: **font-weight: bold**

Example: `<p>The slabs at the ends of letter strokes are called <b>serifs</b>.</p>`

NOTE

It helps me to think about how a screen reader would read the text. If I don't want the word read in a loud, emphatic tone of voice, but it really should be bold, then `b` *may be more appropriate than* `strong`.

Obras protegidas por Direitos de Autor

i

HTML 4.01 definition: Italic

HTML5 definition: Indicates text that is in a different voice or mood than the surrounding text, such as a phrase from another language, a technical term, or thought.

CSS alternative: For italic text, use **font-style**. Example: **font-style: italic**

Example:
```
<p>Simply change the font and <i>Voila!</i>, a new personality!</p>
```

s

HTML 4.01 definition: Strike-through text

HTML5 definition: Indicates text that is incorrect.

CSS Property: To put a line through a text selection, use **text-decoration**. Example: **text-decoration: line-through**

Example:
```
<p>Scala Sans was designed by <s>Eric Gill</s> Martin Majoor.</p>
```

u

HTML 4.01 definition: Underline

HTML5 definition: There are a few instances when underlining has semantic significance, such as underlining a formal name in Chinese or indicating a misspelled word after a spell check. Note that underlined text is easily confused as a link and should generally be avoided except for a few niche cases.

CSS Property: For underlined text, use **text-decoration**. Example: **text-decoration: underline**

Example:
```
<p>New York subway signage is set in <u>Halvetica</u>.</p>
```

small

HTML 4.01 definition: Renders in font smaller than the surrounding text

HTML5 definition: Indicates an addendum or side note to the main text, such as the legal "small print" at the bottom of a document.

CSS Property: To make text smaller, use **font-size**. Example: **font-size: 80%**

Example:
```
<p><small>(This font is free for personal and commercial use.)</small></p>
```


Obras protegidas por Direitos de Autor

The slabs at the ends of letter strokes are called **serifs**.

Simply change the font and *Voila!*, a new personality!

Scala Sans was designed by ~~Erie Gill~~ Martin Majoor.

New York subway signage is set in Halvetica.

(This font is free for personal and commercial use.)

Figure 5-10. The default rendering of `b`, `i`, `s`, `u`, *and* `small` *elements.*

Short quotations

Use the quotation (`q`) element to mark up short quotations, such as "To be or not to be," in the flow of text, as shown in this example (Figure 5-11).

```
Matthew Carter says, <q>Our alphabet hasn't changed in eons.</q>
```

`<q>...</q>`

Short inline quotation

According to the HTML spec, browsers should add quotation marks around `q` elements automatically, so you don't need to include them in the source document. And for the most part they do, with the exception of Internet Explorer versions 7 and earlier. Fortunately, as of this writing, those browsers make up only 5–8% of browser usage, and it's sure to be significantly less by the time you read this. If you are concerned about a small percentage of users seeing quotations without their marks, stick with using quotation marks in your source, a fine alternative.

Matthew Carter says, "Our alphabet hasn't changed in eons."

Figure 5-11. Nearly all browsers add quotation marks automatically around `q` *elements.*

REMINDER

Nesting Elements

You can apply two elements to a string of text (for example, a phrase that is both a quote and in another language), but be sure they are nested properly. That means the inner element, including its closing tag, must be completely contained within the outer element, and not overlap.

```
<q><i>Je ne sais pas.</i></q>
```

Abbreviations and acronyms

Marking up acronyms and abbreviations with the `abbr` element provides useful information for search engines, screen readers, and other devices. Abbreviations are shortened versions of a word ending in a period (Conn. for Connecticut, for example). Acronyms are abbreviations formed by the first letters of the words in a phrase (such as WWW or USA). The `title` attribute provides the long version of the shortened term, as shown in this example:

```
<abbr title="Points">pts.</abbr>
<abbr title="American Type Founders">ATF</abbr>
```

`<abbr>...</abbr>`

Abbreviation or acronym

NOTE

In HTML 4.01, there was an `acronym` *element especially for acronyms, but it has been made obsolete in HTML5 in favor of using the* `abbr` *for both.*

Obras protegidas por Direitos de Autor

Citations

`<cite>...</cite>`
Citation

The **cite** element is used to identify a reference to another document, such as a book, magazine, article title, and so on. Citations are typically rendered in italic text by default. Here's an example:

```
<p>Passages of this article were inspired by <cite>The Complete Manual
of Typography</cite> by James Felici.</p>
```

Defining terms

`<dfn>...</dfn>`
Defining term

It is common to point out the first and defining instance of a word in a document in some fashion. In this book, defining terms are set in blue text. In HTML, you can identify them with the **dfn** element and format them visually using style sheets.

```
<p><dfn>Script typefaces</dfn> are based on handwriting.</p>
```

Program code elements

`<code>...</code>`
Code

`<var>...</var>`
Variable

`<samp>...</samp>`
Program sample

`<kbd>...</kbd>`
User-entered keyboard strokes

A number of inline elements are used for describing the parts of technical documents, such as code (**code**), variables (**var**), program samples (**samp**), and user-entered keyboard strokes (**kbd**). For me, it's a quaint reminder of HTML's origins in the scientific world (Tim Berners-Lee developed HTML to share documents at the CERN particle physics lab in 1989).

Code, sample, and keyboard elements typically render in a constant-width (also called monospace) font such as Courier by default. Variables usually render in italics.

Subscript and superscript

`<sub>...</sub>`
Subscript

`<sup>...</sup>`
Superscript

The subscript (**sub**) and superscript (**sup**) elements cause the selected text to display in a smaller size, positioned slightly below (**sub**) or above (**sup**) the baseline. These elements may be helpful for indicating chemical formulas or mathematical equations.

Figure 5-12 shows how these examples of subscript and superscript typically render in a browser.

```
<p>H<sub>2</sub>0</p>

<p>E=MC<sup>2</sup></p>
```

H_20 $E=MC^2$

Figure 5-12. Subscript and superscript

Obras protegidas por Direitos de Autor

Highlighted text

The new **mark** element indicates a word that may be considered especially relevant to the reader. One might use it to call out a search term in a page of results, to manually call attention to a passage of text, or to indicate the current page in a series. Some designers (and browsers) give marked text a light colored background as though it were marked with a highlighter marker, as shown in Figure 5-13.

```
<p> ... PART I. ADMINISTRATION OF THE GOVERNMENT. TITLE IX.
TAXATION. CHAPTER 65C. MASS. <mark>ESTATE TAX</mark>. Chapter 65C:
Sect. 2. Computation of <mark>estate tax</mark>.</p>
```

... PART I. ADMINISTRATION OF THE GOVERNMENT. TITLE IX. TAXATION. CHAPTER 65C. MASS. ESTATE TAX. Chapter 65C: Sect. 2. Computation of estate tax.

Figure 5-13. Search terms are marked as mark elements and given a yellow background with a style sheet so they are easier for the reader to find.

`<mark>...</mark>`

Contextually relevant text

NEW IN HTML5

SUPPORT ALERT

The **mark** *element is not supported in Internet Explorer versions 8 and earlier (see the sidebar "HTML5 Support in Internet Explorer," earlier in this chapter, for a workaround). Older versions of Firefox and Safari (prior to 3.6 and 4, respectively) do not support it according to the spec, but do allow you to apply styles to it.*

Times and machine-readable information

When we look at the phrase "noon on November 4," we know that it is a date and a time. But the context might not be so obvious to a computer program. The **time** element allows us to mark up dates and times in a way that is comfortable for a human to read, but also encoded in a standardized way that computers can use. The content of the element presents the information to people, and the **datetime** attribute presents the same information in a machine-readable way.

The **time** element indicates dates, times, or date-time combos. It might be used to pass the date and time information to an application, such as saving an event to a personal calendar. It might be used by search engines to find the most recently published articles. Or it could be used to restyle time information into an alternate format (e.g., changing 18:00 to 6 p.m.).

The **datetime** attribute specifies the date and/or time information in a standardized time format illustrated in Figure 5-14. It begins with the date (year, month, day), followed by the letter T to indicate time, listed in hours, minutes, seconds (optional), and milliseconds (also optional). Finally, the time zone is indicated by the number of hours behind (–) or ahead (+) of Greenwich Mean Time (GMT). For example, "–05:00" indicates the Eastern Standard time zone, which is five hours behind GMT.

`<time>...</time>`

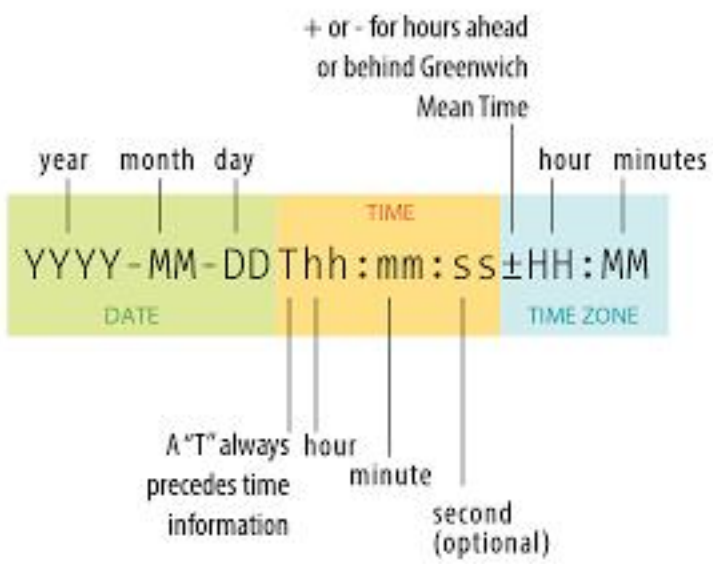
Time data

NEW IN HTML5

NOTE

The **time** *element is not intended for marking up times for which a precise time or date cannot be established, such as "the end of last year" or "the turn of the century."*

Figure 5-14. Standardized date and time syntax.



Example:

3pm PST on December 25, 2012

```
2012-12-25T15:00-8:00
```


Obras protegidas por Direitos de Autor

The WHATWG HTML specification includes a **pubdate** attribute for indicating that the time is the publication date of a document, as shown in this example. The **pubdate** attribute is not included in the W3C HTML5 spec as of this writing, but it may be included at a later date if it becomes widely used.

```
Written by Jennifer Robbins (<time datetime="2012-09-01T 20:00-05:00"
pubdate>September 1, 2012, 8pm EST</time>)
```

`<data>...</data>`
Machine-readable data
WHATWG ONLY

The WHATWG also includes the **data** element for helping computers make sense of content, which can be used for all sorts of data, including dates, times, measurements, weights, and so on. It uses the **value** attribute for the machine-readable information. Here are a couple of examples:

```
<data value="12">Twelve</data>
<data value="2011-11-12">Last Saturday</data>
```

SUPPORT ALERT

Both `time` *and* `data` *are new elements and are not universally supported as of this writing. However, you can apply styles to them and they will be recognized by browsers other than IE8 and earlier.*

I'm not going to go into more detail on the **data** element, because as of this writing, the powers that be are still discussing exactly how it should work, and the W3C has not adopted it for the HTML5 spec. Also, as a beginner, you are unlikely to be dealing with machine-readable data yet anyway. But still, it is interesting to see how markup can be used to provide usable information to computer programs and scripts as well as to your fellow humans.

Inserted and deleted text

`<ins>...</ins>`
Inserted text

`<del>...</del>`
Deleted text

The **ins** and **del** elements are used to mark up edits indicating parts of a document that have been inserted or deleted (respectively). These elements rely on style rules for presentation (i.e., there is no dependable browser default). Both the **ins** and **del** elements can contain either inline or block elements, depending on what type of content they contain.

```
Chief Executive Officer: <del title="retired">Peter Pan</del><ins>Pippi
Longstockings</ins>
```

Adding Breaks

Line breaks

`<br>`
Line break

Occasionally, you may need to add a line break within the flow of text. We've seen how browsers ignore line breaks in the source document, so we need a specific directive to tell the browser to "add a line break here."

The inline line break element (**br**) does exactly that. The **br** element could be used to break up lines of addresses or poetry. It is an empty element, which means it does not have content. Just add the **br** element (**
** in XHTML) in the flow of text where you want a break to occur, as shown here and in Figure 5-15.

```
<p>So much depends <br>upon <br><br>a red wheel <br>barrow</p>
```

So much depends
upon

a red wheel
barrow

Figure 5-15. Line breaks are inserted at each `br` *element.*

Obras protegidas por Direitos de Autor

Accommodating Non-Western Languages

Because the Web is "world-wide," there are a few elements designed to address the needs of non-Western languages.

Changing direction

The **bdo** (bidirectional override) element allows a phrase in a right-to-left (**rtl**) reading language (such as Hebrew or Arabic) to be included in a left-to-right (**ltr**) reading flow, or vice versa.

```
This is how you write Shalom: <bdo dir="rtl">&#x0
5E9;&#x05DC;&#x05D5;&#x05DD;</bdo>
```

The **bdi** (bidirectional isolation) element is similar, but it is used to isolate a selection that *might* read in a different direction, such as a name or comment added by a user.

Hints for East Asian languages

HTML5 also includes the **ruby**, **rt**, and **rp** elements used to add ruby annotation to East Asian languages. Ruby annotations are little notes that typically appear above ideographs and provide pronunciation clues or translations. Within the **ruby** element, the **rt** element indicates the helpful ruby text. Browsers that support ruby text typically display it in a smaller font above the main text. As a backup for browsers that don't support ruby, you can put the ruby text in parentheses, each marked with the **rp** element. Non-supporting browsers display all the text on the same line, with the ruby in parentheses. Supporting browsers ignore the content of the **rp** elements and display only the **rt** text above the glyphs. The ruby system has spotty browser support as of this writing.

```
<ruby>
 字<rp>(</rp><rt>han</rt><rp>)</rp>
 汉<rp>(</rp><rt>zi</rt><rp>)</rp>
</ruby>
```

This example was taken from the HTML5 Working Draft at whatwg.com, used with permission under an MIT License.

Unfortunately, the **br** element is easily abused (see the warning). Consider whether using the CSS **white-space** property (introduced in Chapter 12, Formatting Text) might be a better alternative for maintaining line breaks from your source without extra markup.

Word breaks

`<wbr>`

Word break

The word break (**wbr**) element lets you mark the place where a word should break if it needs to (a "line break opportunity" according to the spec). It takes some of the guesswork away from the browser and allows authors to specify the best spot for the word to be split over two lines. Keep in mind that the word breaks at the **wbr** element only if it needs to (Figure 5-16). If there is enough room, the word stays in one piece. Browsers have supported this element for a long time, but it has recently been incorporated into the HTML standard.

```
<p>The biggest word you've ever heard and this is how it goes:
<em>supercali<wbr>fragilistic<wbr>expialidocious</em>!</p>
```

The biggest word you've ever heard and this is how it goes: *supercalifragilistic expialidocious*!

Figure 5-16. When there is not enough room for a word to fit on a line, it will break at the location of the **wbr** *element.*

WARNING

Be careful that you aren't using **br** *elements to force breaks into text that really ought to be a list. For example, don't do this:*

```
<p>Times<br>
Georgia<br>
Garamond
</p>
```

If it's a list, use the semantically correct unordered list element instead, and turn off the bullets with style sheets.

```
<ul>
  <li>Times</li>
  <li>Georgia</li>
  <li>Garamond</li>
</ul>
```


Obras protegidas por Direitos de Autor

exercise 5-2 | Identifying inline elements

This little post for the Black Goose Bistro blog will give you an opportunity to identify and mark up a variety of inline elements. See if you can find phrases to mark up accurately with the following elements:

`b` `br` `cite` `dfn` `em` `i` `q` `small` `time`

Because markup is always somewhat subjective, your resulting markup may not look exactly like the example in Appendix A, but there is an opportunity to use all of the elements listed above in the article. For extra credit, there is a phrase that should have two elements applied to it (remember to nest them properly by closing the inner element before you close the outer one).

You can write the tags right on this page. Or, if you want to use a text editor and see the results in a browser, this text file is available online at *www.learningwebdesign.com/4e/materials*. The resulting code appears in Appendix A.

```
<article>

   <header>

   <p>posted by BGB, November 15, 2012</p>

   </header>

<h1>Low and Slow</h1>

<p>This week I am extremely excited about a new cooking technique
called sous vide. In sous vide cooking, you submerge the food
(usually vacuum-sealed in plastic) into a water bath that is
precisely set to the target temperature you want the food to be
cooked to. In his book, Cooking for Geeks, Jeff Potter describes
it as ultra-low-temperature poaching.</p>
<p>Next month, we will be serving Sous Vide Salmon with Dill
Hollandaise. To reserve a seat at the chef table, contact us
before November 30.</p>


<p>blackgoose@example.com
555-336-1800</p>


<p> Warning: Sous vide cooked salmon is not pasteurized. Avoid it
if you are pregnant or have immunity issues.</p>
</article>
```


Obras protegidas por Direitos de Autor

Generic Elements (div and span)

What if none of the elements we've talked about so far accurately describes your content? After all, there are endless types of information in the world, but as you've seen, not all that many semantic elements. Fortunately, HTML provides two generic elements that can be customized to describe your content perfectly. The **div** element indicates a division of content, and **span** indicates a word or phrase for which no text-level element currently exists. The generic elements are given meaning and context with the **id** and **class** attributes, which we'll discuss in a moment.

`<div>...</div>`
Generic block-level element

`<span>...</span>`
Generic inline element

The **div** and **span** elements have no inherent presentation qualities of their own, but you can use style sheets to format them however you like. In fact, generic elements are a primary tool in standards-based web design because they enable authors to accurately describe content and offer plenty of "hooks" for adding style rules. They also allow elements on the page to be accessed and manipulated by JavaScript.

We're going to spend a little time on **div** and **span** (as well as the **id** and **class** attributes) and learn how authors use them to structure content.

Divide it up with a div

The **div** element is used to create a logical grouping of content or elements on the page. It indicates that they belong together in some sort of conceptual unit or should be treated as a unit by CSS or JavaScript. By marking related content as a **div** and giving it a unique **id** identifier or indicating that it is part of a **class**, you give context to the elements in the grouping. Let's look at a few examples of **div** elements.

MARKUP TIP

It is possible to nest **div** elements within other **div** elements, but don't go overboard. You should always strive to keep your markup as simple as possible, so add a **div** element only if it is necessary for logical structure, styling, or scripting.

In this example, a **div** element is used as a container to group an image and two paragraphs into a product "listing."

```
<div class="listing">
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type
design.</p>
</div>
```

By putting those elements in a **div**, I've made it clear that they are conceptually related. It will also allow me to style two **p** elements within listings differently than other paragraphs on the page.

Here is another common use of a **div** used to break a page into sections for layout purposes. In this example, a heading and several paragraphs are enclosed in a **div** and identified as the "news" division.

```
<div id="news">
  <h1>New This Week</h1>
  <p>We've been working on...</p>
  <p>And last but not least,... </p>
</div>
```


Obras protegidas por Direitos de Autor

Now that I have an element known as "news," I could use a style sheet to position it as a column to the right or left of the page. You might be thinking, "Hey Jen, couldn't you use a **section** element for that?" You could! In fact, authors may turn to generic **div**s less now that we have better semantic grouping elements in HTML5.

Get inline with span

A **span** offers the same benefits as the **div** element, except it is used for phrase elements and does not introduce line breaks. Because spans are inline elements, they can only contain text and other inline elements (in other words, you cannot put headings, lists, content-grouping elements, and so on, in a **span**). Let's get right to some examples.

There is no **telephone** element, but we can use a **span** to give meaning to telephone numbers. In this example, each telephone number is marked up as a **span** and classified as "tel":

```
<ul>
  <li>John: <span class="tel">999.8282</span></li>
  <li>Paul: <span class="tel">888.4889</span></li>
  <li>George: <span class="tel">888.1628</span></li>
  <li>Ringo: <span class="tel">999.3220</span></li>
</ul>
```

You can see how the classified spans add meaning to what otherwise might be a random string of digits. As a bonus, the **span** element enables us to apply the same style to phone numbers throughout the site (for example, ensuring line breaks never happen within them, using a CSS **white-space: nowrap** declaration). It makes the information recognizable not only to humans but also to computer programs that know that "tel" is telephone number information. In fact, some values—including "tel"—have been standardized in a markup system known as Microformats that makes web content more useful to software (see the upcoming "Microformats and Metadata" sidebar).

id and class attributes

In the previous examples, we saw the **id** and **class** attributes used to provide context to generic **div** and **span** elements. **id** and **class** have different purposes, however, and it's important to know the difference.

Identification with id

The **id** attribute is used to assign a *unique* identifier to an element in the document. In other words, the value of **id** must be used only once in the document. This makes it useful for assigning a name to a particular element, as though it were a piece of data. See the sidebar "id and class Values" for information on providing values for the **id** attribute.

This example uses the books' ISBN numbers to uniquely identify each listing. No two book listings may share the same **id**.

id and class Values

The values for **id** and **class** attributes should start with a letter (A–Z or a–z) or underscore (although Internet Explorer 6 and earlier have trouble with underscores, so they are generally avoided). They should not contain any character spaces or special characters. Letters, numbers, hyphens, underscores, colons, and periods are OK. Also, the values are case-sensitive, so "sectionB" is not interchangeable with "Sectionb."

Obras protegidas por Direitos de Autor

```
<div id="ISBN0321127307">
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type.
  </p>
</div>

<div id="ISBN0881792063">
  <img src="bringhurst-cover.gif" alt="">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  <p>This lovely, well-written book is concerned foremost
   with creating beautiful typography.</p>
</div>
```

Web authors also use **id** when identifying the various sections of a page. In the following example, there may not be more than one element with the **id** of "main," "links," or "news" in the document.

```
<section id="main">
  <!-- main content elements here -->
</section>

<section id="news">
  <!-- news items here -->
</section>

<aside id="links">
  <!-- list of links here -->
</aside>
```

Not Just for divs

The **id** and **class** attributes may be used with all elements in HTML5, not just **div** and **span**. For example, you could identify an ordered list as "directions" instead of wrapping it in a **div**.

```
<ol id="directions">
  <li>...</li>
  <li>...</li>
  <li>...</li>
</ol>
```

Note that in HTML 4.01, **id** and **class** may be used with all elements except **base**, **basefont**, **head**, **html**, **meta**, **param**, **script**, **style**, and **title**.

Microformats and Metadata

As you've seen, the elements in HTML fall short in describing every type of information. A group of developers decided that if **class** names could be standardized (for example, always using "tel" for telephone numbers), they could establish systems for describing data to make it more useful. This system is called Microformats. Microformats extend the semantics of HTML markup by establishing standard *values* for **id**, **class**, and **rel** attributes rather than creating whole new elements.

There are several Microformat "vocabularies" used to identify things such as contact information (hCard) or calendar items (hCalendar). The Microformats.org site is a good place to learn about them. To give you the general idea, the following example describes the parts of an event using the hCalendar Microformat vocabulary so the browser can automatically add it to your calendar program.

```
<section class="vevent">
 <span class="summary">O'Reilly Emerging
 Technology Conference</span>,
 <time class="dtstart" datetime="20110306">Mar 6
 </time> -
 <time class="dtend" datetime="20110310">10,
 2011</time>
 <div class="location">Manchester Grand Hyatt,
 San Diego, CA</div>
 <a class="url" href="http://events.example.com
 pub/e/403">Permalink</a>
</section>
```

The hCard vocabulary identifies components of typical contact information (stored in vCard format), including: address (**adr**), postal code (**postal-code**), states (**region**), and telephone numbers (**tel**), to name a few. The browser can then use a service to grab the information from the web page and automatically add it to an address book.

There is a lot more to say about Microformats than I can fit in this book. And not only that, but there are two additional, more complex systems for adding metadata to web pages in development at the W3C: RDFa and Microdata. It's not clear how they are all going to shake out in the long run, and I'm thinking that this metadata stuff is more than you want to take on right now anyway. But when you are ready to learn more, WebSitesMadeRight.com has assembled a great big list of introductory articles and tutorials on all three options: *websitesmaderight.com/2011/05/html5-microdata-microformats-and-rdfa-tutorials-and-resources/*.

Obras protegidas por Direitos de Autor

Classification with class

The **class** attribute classifies elements into conceptual groups; therefore, unlike the **id** attribute, multiple elements may share a **class** name. By making elements part of the same class, you can apply styles to all of the labeled elements at once with a single style rule or manipulate them all with a script. Let's start by classifying some elements in the earlier book example. In this first example, I've added **class** attributes to classify each **div** as a "listing" and to classify paragraphs as "descriptions."

```
<div id="ISBN0321127307" class="listing">
  <header>
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  </header>
  <p class="description">A combination of type history and examples of
good and bad type.</p>
</div>

<div id="ISBN0881792063" class="listing">
  <header>
  <img src="bringhurst-cover.gif" alt="">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  </header>
  <p class="description">This lovely, well-written book is concerned
foremost with creating beautiful typography.</p>
</div>
```

TIP

The **id** attribute is used to *identify*.

The **class** attribute is used to *classify*.

Notice how the same element may have both a **class** and an **id**. It is also possible for elements to belong to multiple classes. When there is a list of **class** values, simply separate them with character spaces. In this example, I've classified each **div** as a "book" to set them apart from possible "cd" or "dvd" listings elsewhere in the document.

```
<div id="ISBN0321127307" class="listing book">
  <img src="felici-cover.gif" alt="CMT cover">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p class="description">A combination of type history and examples of
good and bad type.</p>
</div>

<div id="ISBN0881792063" class="listing book">
  <img src="bringhurst-cover.gif" alt="ETS cover">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  <p class="description">This lovely, well-written book is concerned
  foremost with creating beautiful typography.</p>
</div>
```

This should have given you a good introduction to how **div** and **span** elements with **class** and **id** attributes are used to add meaning and organization to documents. We'll work with them even more in the style sheet chapters in Part III.

Obras protegidas por Direitos de Autor

Some Special Characters

There's just one more text-related topic before we close out this chapter.

Some common characters, such as the copyright symbol ©, are not part of the standard set of ASCII characters, which contains only letters, numbers, and a few basic symbols. Other characters, such as the less-than symbol (<), are available, but if you put one in an HTML document, the browser will interpret it as the beginning of a tag.

Characters such as these must be escaped in the source document. Escaping means that instead of typing in the character itself, you represent it by its numeric or named character reference. When the browser sees the character reference, it substitutes the proper character in that spot when the page is displayed.

There are two ways of referring to a specific character: by an assigned numeric value (numeric entity) or using a predefined abbreviated name for the character (called a named entity). All character references begin with an "&" and end with a ";".

Some examples will make this clear. I'd like to add a copyright symbol to my page. The typical Mac keyboard command, Option-G, which works in my word processing program, may not be understood properly by a browser or other software. Instead, I must use the named entity **©** (or its numeric equivalent, **©**) where I want the symbol to appear (Figure 5-17).

```
<p>All content copyright &copy; 2007, Jennifer Robbins</p>
```

or:

```
<p>All content copyright &#169; 2007, Jennifer Robbins</p>
```

HTML defines hundreds of named entities as part of the markup language, which is to say you can't make up your own entity. Table 5-2 lists some commonly used character references. If you'd like to see them all, the complete list of character references has been assembled online by the nice folks at the Web Standards Project at *www.webstandards.org/learn/reference/charts/entities/*.

All content copyright © 2007, Jennifer Robbins

Figure 5-17. The special character is substituted for the character reference when the document is displayed in the browser.

NOTE

In XHTML, every instance of an ampersand must be escaped so that it is not interpreted as the beginning of a character entity, even when it appears in the value of an attribute. For example:

```
<img src="sno.jpg" alt="Sifl
&amp; Olly Show" />
```


Obras protegidas por Direitos de Autor

Non-breaking Spaces

One interesting character to know about is the non-breaking space (` `). Its purpose is to ensure that a line doesn't break between two words. So, for instance, if I mark up my name like this:

```
Jennifer Robbins
```

I can be sure that my first and last names will always stay together on a line.

Table 5-2. Common special characters and their character references

Character	Description	Name	Number
	Non-breaking character space	` `	` `
&	Ampersand	`&`	`&`
'	Apostrophe	`'`	`'`
<	Less-than symbol (useful for displaying markup on a web page)	`<`	`<`
>	Greater-than symbol (useful for displaying markup on a web page)	`>`	`>`
©	Copyright	`©`	`©`
®	Registered trademark	`®`	`®`
™	Trademark	`™`	`™`
£	Pound	`£`	`£`
¥	Yen	`¥`	`¥`
€	Euro	`€`	`€`
–	En-dash	`–`	`–`
—	Em-dash	`—`	`—`
‘	Left curly single quote	`‘`	`‘`
’	Right curly single quote	`’`	`’`
“	Left curly double quote	`“`	`“`
”	Right curly double quote	`”`	`”`
•	Bullet	`•`	`•`
…	Horizontal ellipsis	`…`	`…`

Putting It All Together

TIP

Remember that indenting each hierarchical level in your HTML source consistently makes the document easier to scan and update later.

So far, you've learned how to mark up elements, and you've met all of the HTML elements for adding structure and meaning to text content. Now it's just a matter of practice. Exercise 5-3 gives you an opportunity to try out everything we've covered so far: document structure elements, block elements, inline elements, sectioning elements, and character entities. Have fun!

Obras protegidas por Direitos de Autor

exercise 5-3 | The Black Goose Blog page

Now that you've been introduced to all of the text elements, you can put them to work by marking up the Blog page for the Black Goose Bistro site. The content is shown below (the second post is already marked up with the inline elements from Exercise 5-2). Get the starter text file online at *www.learningwebdesign.com/4e/materials*. The resulting markup is in Appendix A and included in the ***materials*** folder.

Once you have the text file, follow the instructions listed after the copy. The resulting page is shown in Figure 5-18.

```
The Black Goose Blog

Home
Menu
Blog
Contact

Summer Menu Items
posted by BGB, June 15, 2013
Our chef has been busy putting together the
perfect menu for the summer months. Stop by to
try these appetizers and main courses while the
days are still long.

Appetizers
Black bean purses
Spicy black bean and a blend of mexican cheeses
wrapped in sheets of phyllo and baked until
golden. $3.95

Southwestern napoleons with lump crab -- new
item!
Layers of light lump crab meat, bean and corn
salsa, and our handmade flour tortillas. $7.95

Main courses

Shrimp sate kebabs with peanut sauce
Skewers of shrimp marinated in lemongrass, garlic,
and fish sauce then grilled to perfection. Served
with spicy peanut sauce and jasmine rice. $12.95

Jerk rotisserie chicken with fried plantains --
new item!
Tender chicken slow-roasted on the rotisserie,
flavored with spicy and fragrant jerk sauce and
served with fried plantains and fresh mango.
$12.95

Low and Slow
posted by BGB, November 15, 2012
<p>This week I am <em>extremely</em> excited
about a new cooking technique called <dfn><i>sous
vide</i></dfn>. In <i>sous vide</i> cooking,
you submerge the food (usually vacuum-sealed in
plastic) into a water bath that is precisely
set to the target temperature of the food. In
his book, <cite>Cooking for Geeks</cite>, Jeff
Potter describes it as <q>ultra-low-temperature
poaching</q>.</p>

<p>Next month, we will be serving <b>Sous Vide
Salmon with Dill Hollandaise</b>. To reserve
a seat at the chef table, contact us before
November 30.</p>


Location: Baker's Corner, Seekonk, MA
Hours: Tuesday to Saturday, 11am to midnight

All content copyright &copy; 2012, Black Goose
Bistro and Jennifer Robbins
```

Figure 5-18. The finished menu page.

1. Add all the document structure elements first (**html**, **head**, **meta**, **title**, and **body**). Give the document the title "Black Goose Bistro: Blog."

Obras protegidas por Direitos de Autor

2. The first thing we'll do is identify the top-level heading and the list of links as the header for the document by wrapping them in a **header** element (don't forget the closing tag). Within the header, the headline should be an **h1** and the list of links should be an unordered list (**ul**). Don't worry about making the list items links; we'll get to linking in the next chapter. Give the list more meaning by identifying it as the primary navigation for the site (**nav**).
3. This blog page has two posts titled "Summer Menu Items" and "Low and Slow." Mark each one up as an **article**.
4. Now we'll get the first article into shape! Let's create a **header** for this article that contains the heading (**h2** this time because we've moved down in the document hierarchy) and the publication information (**p**). Identify the publication date for the article with the **time** element, just as you did in Exercise 5-2.
5. The content after the header is clearly a simple paragraph. However, the menu has some interesting things going on. It is divided into two conceptual sections (Appetizers and Main Courses), so mark those up as **section** elements. Be careful that the closing section tag (**</section>**) appears before the closing article tag (**</article>**) so the elements are nested correctly and don't overlap. Finally, let's identify the sections with **id** attributes. Name the first one "appetizers" and the second "maincourses."
6. With our sections in place, now we can mark up the content. We're down to **h3** for the headings in each section. Choose the most appropriate list elements to describe the menu item names and their descriptions. Mark up the lists and each item within the lists.
7. Now we can add a few fine details. *Classify* each price as "price" using **span** elements.
8. Two of the dishes are new items. Change the double hyphens to an em-dash character and mark up "new items!" as "strongly important." Classify the title of each new dish as "newitem" (hint: use the existing **dt** element; there is no need to add a **span** this time). This allows us to target menu titles with the "newitem" class and style them differently than other menu items.
9. That takes care of the first article. The second article is already mostly marked up from the previous exercise, but you should mark up the header with the appropriate heading and publication information.
10. So far, so good, right? Now make the remaining content that applies to the whole page a **footer**. Mark each line of content within the footer as a paragraph.
11. Let's give the location and hours information some context by putting them in a **div** named "about." Make the labels "Location" and "Hours" appear on a line by themselves by adding line breaks after them. If you'd like, you could also mark up the hours with the **time** element.
12. Finally, copyright information is typically "small print" on a document, so mark it up accordingly. As the final touch, add a copyright symbol after the word "copyright."

Save the file, name it *bistro_blog.html*, and check your page in a modern browser (remember that IE 8 and earlier won't know what to do with those new HTML5 sectioning elements). How did you do?

Markup tips:

- Choose the element that best fits the meaning of the selected text.
- Don't forget to close elements with closing tags.
- Put all attribute values in quotation marks for clarity.
- "Copy and paste" is your friend when adding the same markup to multiple elements. Just be sure what you copied is correct before you paste it throughout the document.

Test Yourself

Were you paying attention? Here is a rapid-fire set of questions to find out.

1. Add the markup to add a thematic break between these paragraphs.

```
<p>People who know me know that I love to cook.</p>

<p>I've created this site to share some of my favorite
recipes.</p>
```

2. What's the difference between a **blockquote** and a **q** element?

Obras protegidas por Direitos de Autor

3. Which element displays whitespace exactly as it is typed into the source document?

4. What is the difference between a **ul** and an **ol**?

5. How do you remove the bullets from an unordered list? (Be general, not specific.)

6. What element would you use to provide the full name of the W3C (World Wide Web Consortium) in the document? Can you write out the complete markup?

7. What is the difference between a **dl** and a **dt**?

Want More Practice?

Try marking up your own résumé. Start with the raw text and add document structure elements, content grouping elements, and inline elements as we've done in Exercise 5-3. If you don't see an element that matches your information just right, try creating one using a **div** or a **span**.

8. What is the difference between **id** and **class**?

9. What is the difference between an **article** and a **section**?

10. Name and write the characters generated by these character entities:

`—`	_____	`&`	_____
` `	_____	`©`	_____
`•`	_____	`™`	_____

Obras protegidas por Direitos de Autor

Element Review: Text

The following is a summary of the elements we covered in this chapter. New HTML5 elements are indicated by "(5)." The **data** element is included only in the WHATWG HTML version as of this writing.

Page sections

address	author contact information
article (5)	self-contained content
aside (5)	tangential content (sidebar)
footer (5)	related content
header (5)	introductory content
nav (5)	primary navigation
section (5)	conceptually related group of content

Heading content

h1...h6	headings, levels 1 through 6
hgroup	heading group

Grouping content

blockquote	blockquote
div	generic division
figure (5)	related image or resource
figcaption (5)	text description of a figure
hr	paragraph-level thematic break (horizontal rule)
p	paragraph
pre	preformatted text

List elements

dd	definition
dl	definition list
dt	term
li	list item (for **ul** and **ol**)
ol	ordered list
ul	unordered list

Breaks

br	line break
wbr (5)	word break

Phrasing elements

abbr	abbreviation
b	added visual attention (bold)
bdi (5)	possible direction change
bdo	bidirectional override
cite	citation
code	code sample
data (WHATWG)	machine-readable equivalent
del	deleted text
dfn	defining term
em	stress emphasis
i	alternate voice (italic)
ins	inserted text
kbd	keyboard text
mark (5)	highlighted text
q	short inline quotation
ruby (5)	section containing ruby text
rp (5)	parentheses in ruby text
rt (5)	ruby annotations
s	strike-through; incorrect text
samp	sample output
small	annotation; "small print"
span	generic phrase of text
strong	strong importance
sub	subscript
sup	superscript
time (5)	machine-readable time data
u	added attention (underline)
var	variable

Obras protegidas por Direitos de Autor

ADDING LINKS

If you're creating a page for the Web, chances are you'll want it to point to other web pages and resources, whether on your own site or someone else's. Linking, after all, is what the Web is all about. In this chapter, we'll look at the markup that makes links work—links to other sites, to your own site, and within a page. There is one element that makes linking possible: the anchor (**a**).

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

Anchor element (hypertext link)

To make a selection of text a link, simply wrap it in opening and closing **<a>...** tags and use the **href** attribute to provide the URL of the target page. The content of the anchor element becomes the hypertext link. Here is an example that creates a link to the O'Reilly Media website:

```
<a href="http://www.oreilly.com">Go to the O'Reilly Media site</a>
```

To make an image a link, simply put the **img** element in the anchor element:

```
<a href="http://www.oreilly.com"><img src="orm.gif" alt="O'Reilly
tarsier logo"></a>
```

Nearly all graphical browsers display linked text as blue and underlined by default. Some older browsers put a blue border around linked images, but most current ones do not. Visited links generally display in purple. Users can change these colors in their browser preferences, and, of course, you can change the appearance of links for your sites using style sheets. I'll show you how in Chapter 13, Colors and Backgrounds.

WARNING

One word of caution: if you choose to change your link colors, keep them consistent throughout your site so as not to confuse your users.

When a user clicks or taps on the linked text or image, the page you specify in the anchor element loads in the browser window. The linked image markup sample shown previously might look like Figure 6-1.

IN THIS CHAPTER

Making links to external pages

Making relative links to documents on your own server

Linking to a specific point in a page

Adding "mailto" and "tel" links

Targeting new windows

AT A GLANCE

Anchor Syntax

The simplified structure of the anchor element is:

```
<a href="url">linked text
or element</a>
```


Obras protegidas por Direitos de Autor

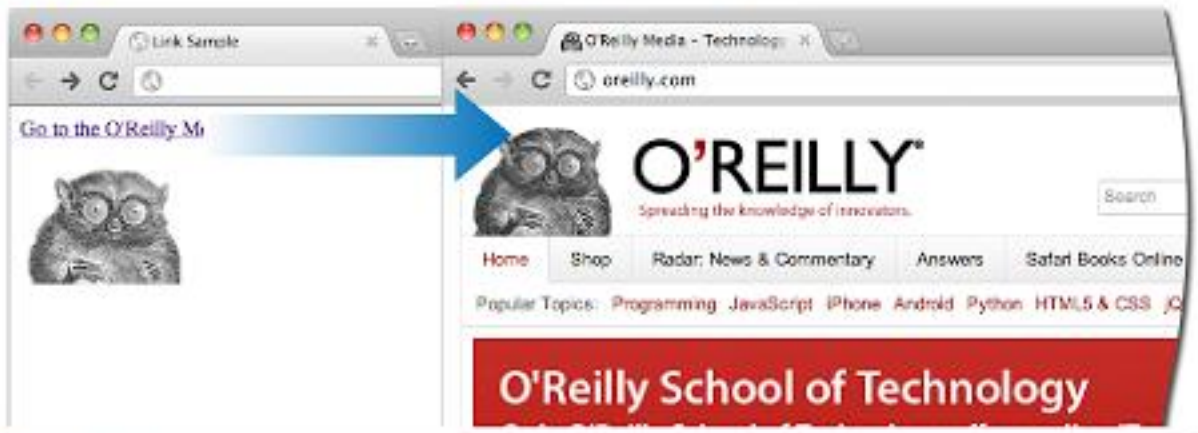


Figure 6-1. When a user clicks or taps on the linked text or image, the page you specified in the anchor element loads in the browser window.

In HTML5, you can put any element in an a element—even block elements!

Starting in HTML5, you can put any element in an **a** element—even block elements! In the HTML 4.01 spec and earlier, the anchor element could be used for inline content only.

The href Attribute

You'll need to tell the browser which document to link to, right? The **href** (hypertext reference) attribute provides the address of the page or resource (its URL) to the browser. The URL must always appear in quotation marks. Most of the time you'll point to other HTML documents; however, you can also point to other web resources, such as images, audio, and video files.

Because there's not much to slapping anchor tags around some content, the real trick to linking comes in getting the URL correct.

There are two ways to specify the URL:

- Absolute URLs provide the full URL for the document, including the protocol (**http://**), the domain name, and the pathname as necessary. You need to use an absolute URL when pointing to a document out on the Web (i.e., not on your own server).

  Example: `href="http://www.oreilly.com/"`

  Sometimes, when the page you're linking to has a long URL pathname, the link can end up looking pretty confusing (Figure 6-2). Just keep in mind that the structure is still a simple container element with one attribute. Don't let the pathname intimidate you.

- Relative URLs describe the pathname to a file *relative* to the current document. Relative URLs can be used when you are linking to another document on your own site (i.e., on the same server). It doesn't require the protocol or domain name—just the pathname.

  Example: `href="recipes/index.html"`

In this chapter, we'll add links using absolute and relative URLs to my cooking website, Jen's Kitchen (see Figure 6-3). Absolute URLs are easy, so let's get them out of the way first.

URL Versus URI

The W3C and the development community are moving away from the term URL (Uniform Resource Locator) and toward the more generic and technically accurate URI (Uniform Resource Identifier). On the street and even on the job, however, you're still likely to hear URL.

Here's the skinny on URL versus URI: a URL is one type of a URI that identifies the resource by its location (the L in URL) on the network. The other type of URI is a URN that identifies the resource by name or namespace (the N in URN).

Because it is more familiar, I will be sticking with URL throughout the discussions in this chapter. Just know that URLs are a subset of URIs, and the terms are often used interchangeably.

If you like to geek out on this kind of thing, I refer you to the URI Wikipedia entry: *en.wikipedia.org/wiki/Uniform_resource_identifier.*

Obras protegidas por Direitos de Autor

Figure 6-2. An example of a long URL. Although it may make the anchor tag look confusing, the structure is the same.

MARKUP TIP

URL Wrangling

If you are linking to a page with a long URL, it is helpful to copy the URL from the location toolbar in your browser and paste it into your document. That way, you avoid mistyping a single character and breaking the whole link.

Linking to Pages on the Web

Many times, you'll want to create a link to a page that you've found on the Web. This is known as an "external" link because it is going to a page outside of your own server or site. To make an external link, you need to provide the absolute URL, beginning with **http://** (the protocol). This tells the browser, "Go out on the Web and get the following document."

I want to add some external links to the Jen's Kitchen home page (Figure 6-3). First, I'll link the list item "The Food Network" to the site *www.foodnetwork.com*. I marked up the link text in an anchor element by adding opening and closing anchor tags. Notice that I've added the anchor tags *inside* the list item (**li**) element. That's because only **li** elements are permitted to be children of a **ul** element; placing an **a** element directly inside the **ul** would be invalid HTML.

```
<li><a>The Food Network</a></li>
```

Next, I add the **href** attribute with the complete URL for the site.

```
<li><a href="http://www.foodnetwork.com">The Food Network</a></li>
```

And *voilá*! That's all there is to it. Now "The Food Network" will appear as a link and will take my visitors to that site when they click or tap it.

TRY IT

Work Along with Jen's Kitchen

Figure 6-3. The Jen's Kitchen page

All the files for the Jen's Kitchen website are available online at *www.learningwebdesign.com/4e/materials*. Download the entire directory, making sure not to change the way its contents are organized.

The resulting markup for all of the exercises is provided in Appendix A.

The pages aren't much to look at, but they will give you a chance to develop your linking skills.

exercise 6-1 | Make an external link

Open the file *index.html* from the *jenskitchen* folder. Make the list item "Epicurious" link to its web page at *www.epicurious.com*, following my example.

```
<ul>
  <li><a href="http://www.foodnetwork.com/">The Food Network</a>
</li>
  <li>Epicurious</li>
</ul>
```

When you are done, you can save *index.html* and open it in a browser. If you have an Internet connection, you can click on your new link and go to the Epicurious site. If the link doesn't take you there, go back and make sure that you didn't miss anything in the markup.

Obras protegidas por Direitos de Autor

Linking Within Your Own Site

NOTE

On PCs and Macs, files are organized into "folders," but in the web development world, it is more common to refer to the equivalent and more technical term, "directory." A folder is just a directory with a cute icon.

A large portion of the linking you'll do will be between pages of your own site: from the home page to section pages, from section pages to content pages, and so on. In these cases, you can use a relative URL—one that calls for a page on your own server.

Without "http://", the browser looks on the current server for the linked document. A pathname, the notation used to point to a particular file or directory, tells the browser where to find the file. Web pathnames follow the Unix convention of separating directory and filenames with forward slashes (/). A relative pathname describes how to get to the linked document starting from the location of the current document.

Relative pathnames can get a bit tricky. In my teaching experience, nothing stumps beginners like writing relative pathnames, so we'll take it one step at a time. There are exercises along the way that I recommend you do as we go along.

All of the pathname examples in this section are based on the structure of the Jen's Kitchen site shown in Figure 6-4. When you diagram the structure of the directories for a site, it generally ends up looking like an inverted tree with the root directory at the top of the hierarchy. For the Jen's Kitchen site, the root directory is named *jenskitchen*. For another way to look at it, there is also a view of the directory and subdirectories as they appear in the Finder on my Mac (Windows users see one directory at a time).

Important Pathname Don'ts

When you are writing relative pathnames, it is critical that you follow these rules to avoid common errors:

- Don't use backslashes (\). Web URL pathnames use forward slashes (/) only.
- Don't start with the drive name (D:, C:, etc.). Although your pages will link to each other successfully while they are on your own computer, once they are uploaded to the web server, the drive name is irrelevant and will break your links.
- Don't start with file://. This also indicates that the file is local and causes the link to break when it is on the server.

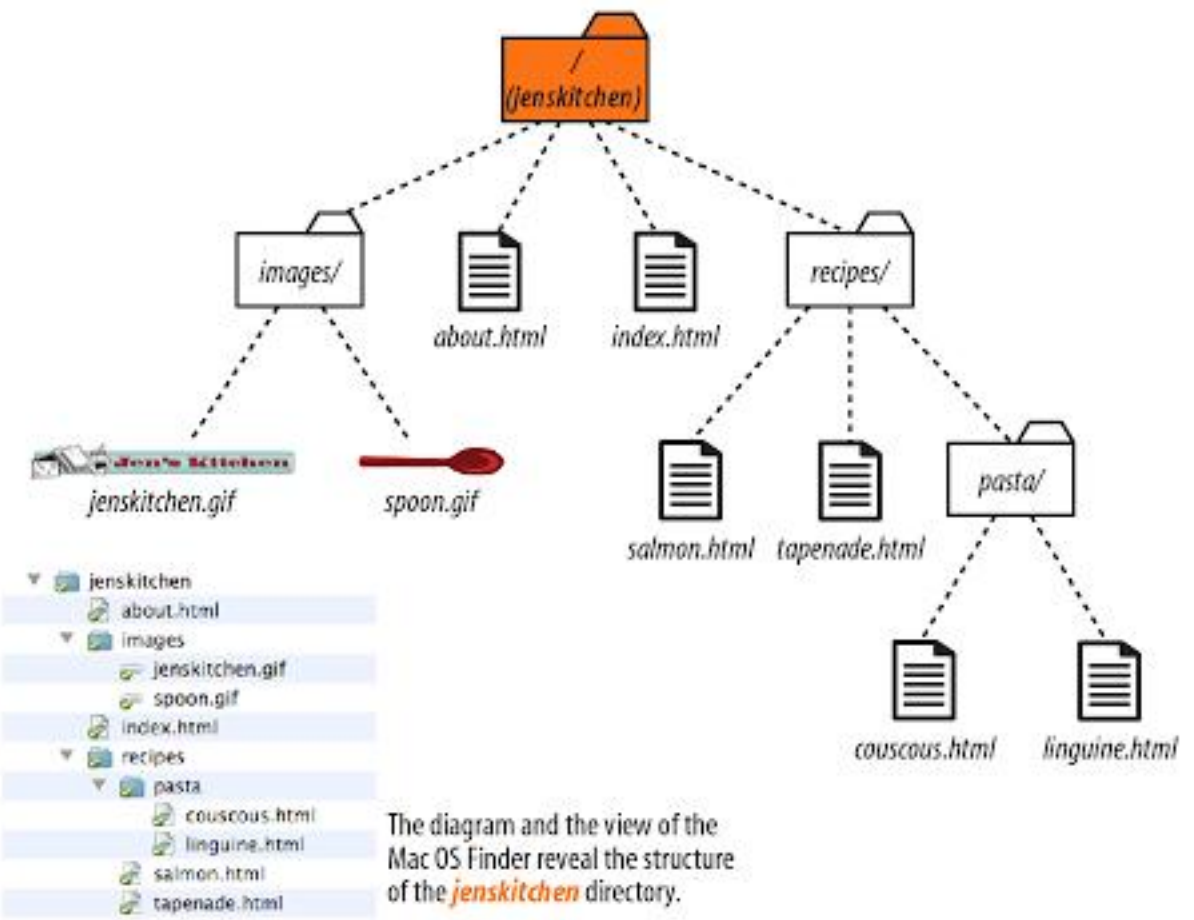


Figure 6-4. A diagram of the jenskitchen site structure.

Obras protegidas por Direitos de Autor

Linking within a directory

The most straightforward relative URL points to another file within the same directory. When you link to a file in the same directory, you only need to provide the name of the file (its filename). When the URL is a single filename, the server looks in the current directory (that is, the directory that contains the document with the link) for the file.

In this example, I want to make a link from my home page (*index.html*) to a general information page (*about.html*). Both files are in the same directory (*jenskitchen*). So from my home page, I can make a link to the information page by simply providing its filename in the URL (Figure 6-5):

```
<a href="about.html">About the site...</a>
```

A link to just the filename indicates the linked file is in the same directory as the current document.

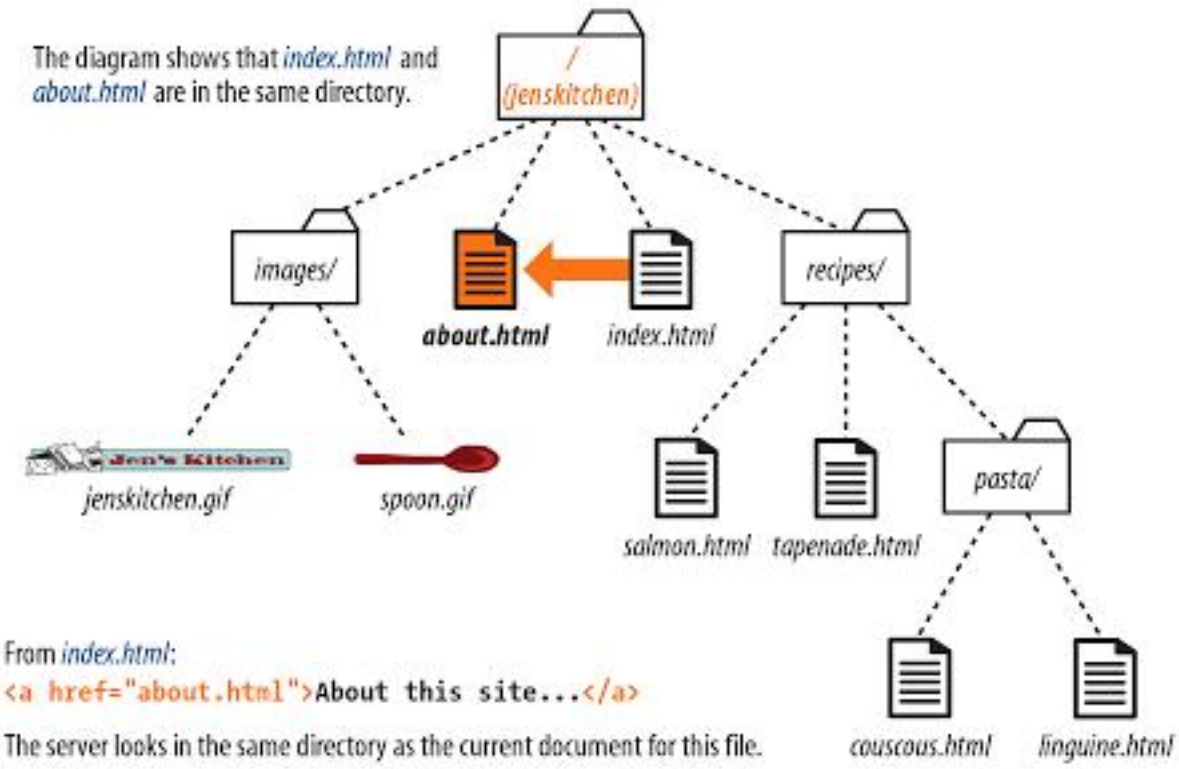


Figure 6-5. Writing a relative URL to another document in the same directory.

exercise 6-2 | Link in the same directory

Open the file ***about.html*** from the ***jenskitchen*** folder. Make the paragraph "Back to the home page" at the bottom of the page link back to ***index.html***. The anchor element should be contained in the `p` element.

```
<p>Back to the home page</p>
```

When you are done, you can save ***about.html*** and open it in a browser. You don't need an Internet connection to test links locally (that is, on your own computer). Clicking on the link should take you back to the home page.

Obras protegidas por Direitos de Autor

Linking to a lower directory

But what if the files aren't in the same directory? You have to give the browser directions by including the pathname in the URL. Let's see how this works.

Getting back to our example, my recipe files are stored in a subdirectory called *recipes*. I want to make a link from *index.html* to a file in the *recipes* directory called *salmon.html*. The pathname in the URL tells the browser to look in the current directory for a directory called *recipes*, and then look for the file *salmon.html* (Figure 6-6):

```
<li><a href="recipes/salmon.html">Garlic Salmon</a></li>
```

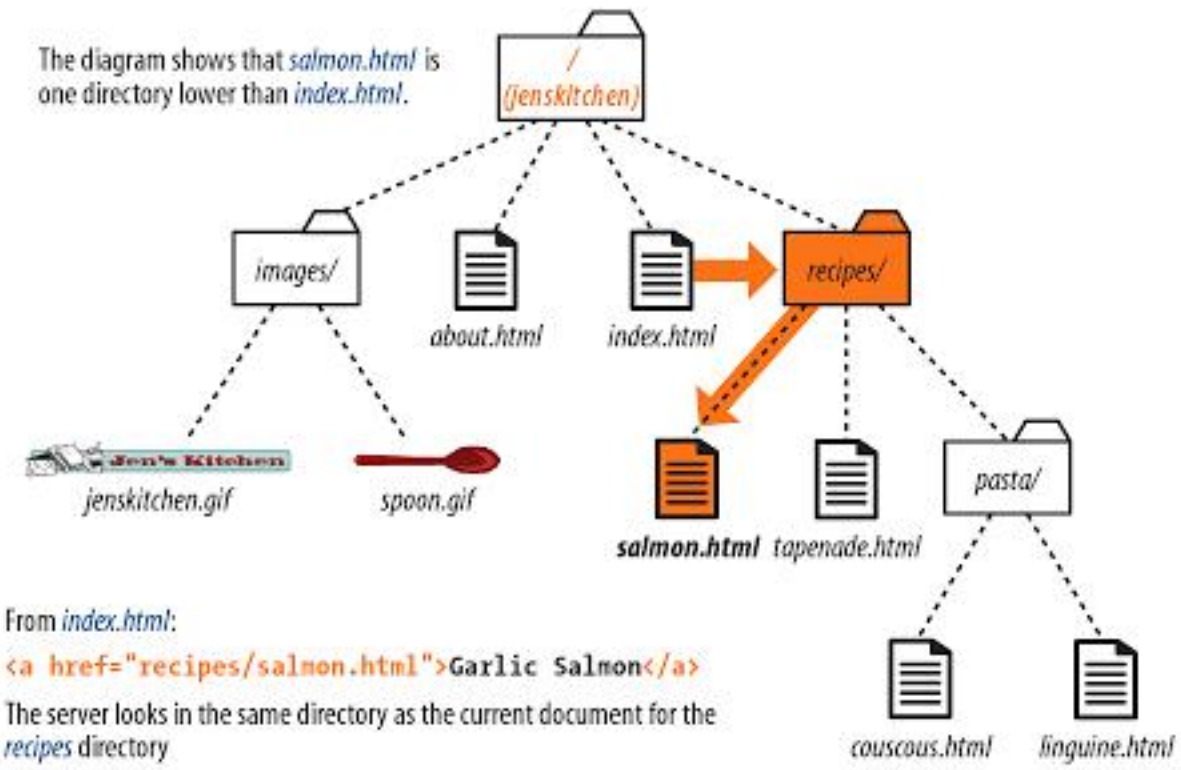


Figure 6-6. Writing a relative URL to a document that is one directory level lower than the current document.

exercise 6-3 | Link one directory down

Open the file ***index.html*** from the ***jenskitchen*** folder. Make the list item "Tapenade (Olive Spread)" link to the file ***tapenade.html*** in the ***recipes*** directory. Remember to nest the elements correctly.

```
<li>Tapenade (Olive Spread)</li>
```

When you are done, you can save ***index.html*** and open it in a browser. You should be able to click your new link and see the recipe page for tapenade. If not, make sure that your markup is correct and that the directory structure for ***jenskitchen*** matches the examples.

Obras protegidas por Direitos de Autor

Now let's link down to the file called *couscous.html*, which is located in the *pasta* subdirectory. All we need to do is provide the directions through two subdirectories (*recipes*, then *pasta*) to *couscous.html* (Figure 6-7):

```
<li><a href="recipes/pasta/couscous.html">Couscous with Peas and Mint
</a></li>
```

When linking to a file in a lower directory, the pathname must contain the names of the subdirectories you go through to get to the file.

Directories are separated by forward slashes. The resulting anchor tag tells the browser, "Look in the current directory for a directory called *recipes*. There you'll find another directory called *pasta*, and in there is the file I'd like to link to, *couscous.html*."

Now that we've done two directory levels, you should get the idea of how pathnames are assembled. This same method applies for relative pathnames that drill down through any number of directories. Just start with the name of the directory that is in the same location as the current file, and follow each directory name with a slash until you get to the linked filename.

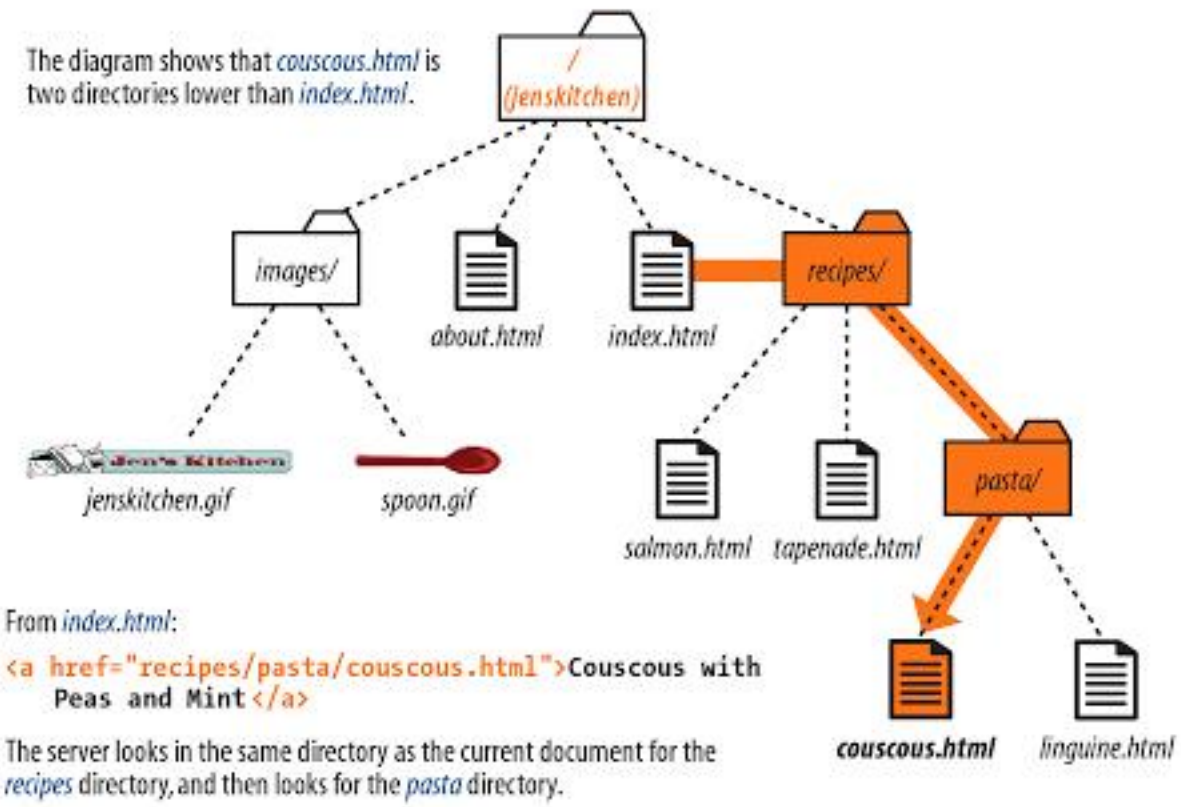


Figure 6-7. Writing a relative URL to a document that is two directory levels lower than the current document.

exercise 6-4 | Link two directories down

Open the file ***index.html*** from the ***jenskitchen*** folder. Make the list item "Linguine with Clam Sauce" link to the file ***linguine.html*** in the ***pasta*** directory.

```
<li>Linguine with Clam Sauce</li>
```

When you are done, you can save ***index.html*** and open it in a browser. Click on the new link to get the delicious recipe.

Obras protegidas por Direitos de Autor

Linking to a higher directory

Each ../ at the beginning of the pathname tells the browser to go up one directory level to look for the file.

So far, so good, right? Here comes the tricky part. This time we're going to go in the other direction and make a link from the salmon recipe page back to the home page, which is one directory level up.

In Unix, there is a pathname convention just for this purpose, the "dot-dot-slash" (`../`). When you begin a pathname with `../`, it's the same as telling the browser "back up one directory level" and then follow the path to the specified file. If you are familiar with browsing files on your desktop, it is helpful to know that a "../" has the same effect as clicking the "Up" button in Windows Explorer or the left-arrow button in the Finder on Mac OS X.

Let's start by making a link from *salmon.html* back to the home page (*index.html*). Because *salmon.html* is in the *recipes* subdirectory, we need to back up a level to *jenskitchen* to find *index.html*. This pathname tells the browser to "go up one level," then look in that directory for *index.html* (Figure 6-8):

```
<p><a href="../index.html">[Back to home page]</a></p>
```

Note that we don't need to write out the name of the higher directory (*jenskitchen*) in the pathname. The `../` stands in for it.

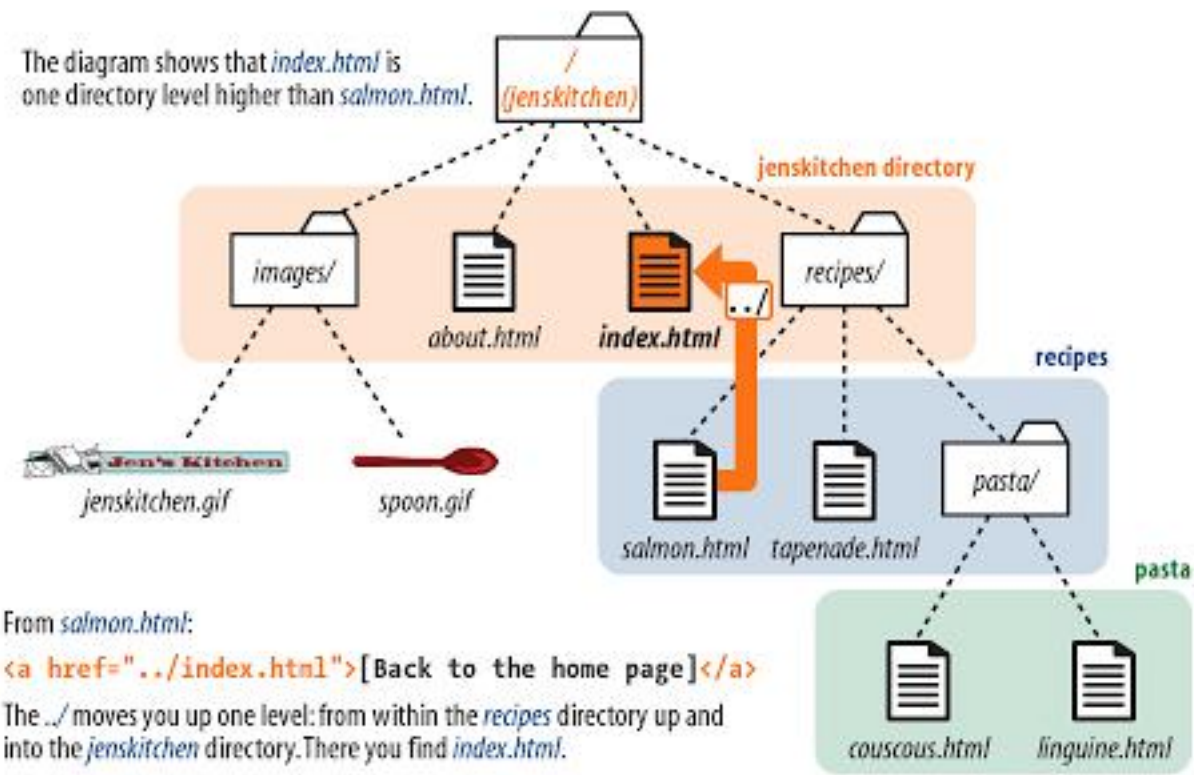


Figure 6-8. Writing a relative URL to a document that is one directory level higher than the current document.

exercise 6-5 | Link to a higher directory

Open the file ***tapenade.html*** from the ***recipes*** directory. At the bottom of the page, you'll find this paragraph:

```
<p>[Back to the home page]</p>
```

Using the notation described in this section, make this text link back to the home page (***index.html***), located one directory level up.

Obras protegidas por Direitos de Autor

But how about linking back to the home page from *couscous.html*? Can you guess how you'd back your way out of two directory levels? Simple: just use the dot-dot-slash twice (Figure 6-9).

A link on the *couscous.html* page back to the home page (*index.html*) would look like this:

```
<p><a href="../../index.html">[Back to home page]</a></p>
```

The first `../` backs up to the *recipes* directory; the second `../` backs up to the top-level directory where *index.html* can be found. Again, there is no need to write out the directory names; the `../` does it all.

NOTE

I confess to still sometimes silently chanting "go-up-a-level, go-up-a-level" for each `../` when trying to decipher a complicated relative URL. It helps me sort things out.

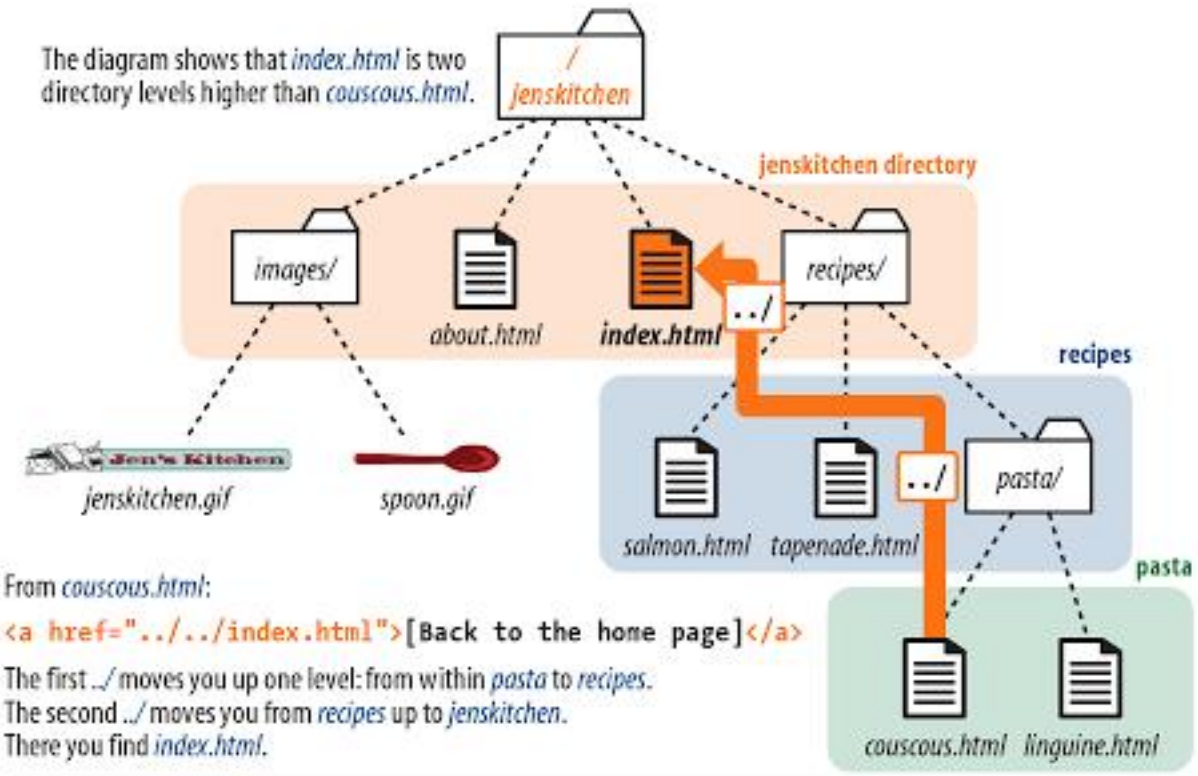


Figure 6-9. Writing a relative URL to a document that is two directory levels higher than the current document.

exercise 6-6 | Link up two directory levels

OK, now it's your turn to give it a try. Open the file ***linguine.html*** and make the last paragraph link back to the home page using `../../` as I have done.

```
<p>[Back to the home page]</p>
```

When you are done, save the file and open it in a browser. You should be able to link to the home page.

Obras protegidas por Direitos de Autor

Site root relative pathnames

All websites have a root directory, which is the directory that contains all the directories and files for the site. So far, all of the pathnames we've looked at are relative to the document with the link. Another way to write a relative pathname is to start at the root directory and list the subdirectory names until you get to the file you want to link to. This kind of pathname is known as site root relative.

Site root relative links are generally preferred due to their flexibility.

In the Unix pathname convention, a forward slash (/) at the start of the pathname indicates the path begins at the root directory. The site root relative pathname in the following link reads, "Go to the very top-level directory for this site, open the *recipes* directory, then find the *salmon.html* file" (Figure 6-10):

```
<a href="/recipes/salmon.html">Garlic Salmon</a>
```

Note that you don't need to (and you shouldn't) write the name of the root directory (*jenskitchen*) in the path—just start it with a forward slash (/), and the browser will look in the top-level directory relative to the current file. From there, just specify the directories the browser should look in.

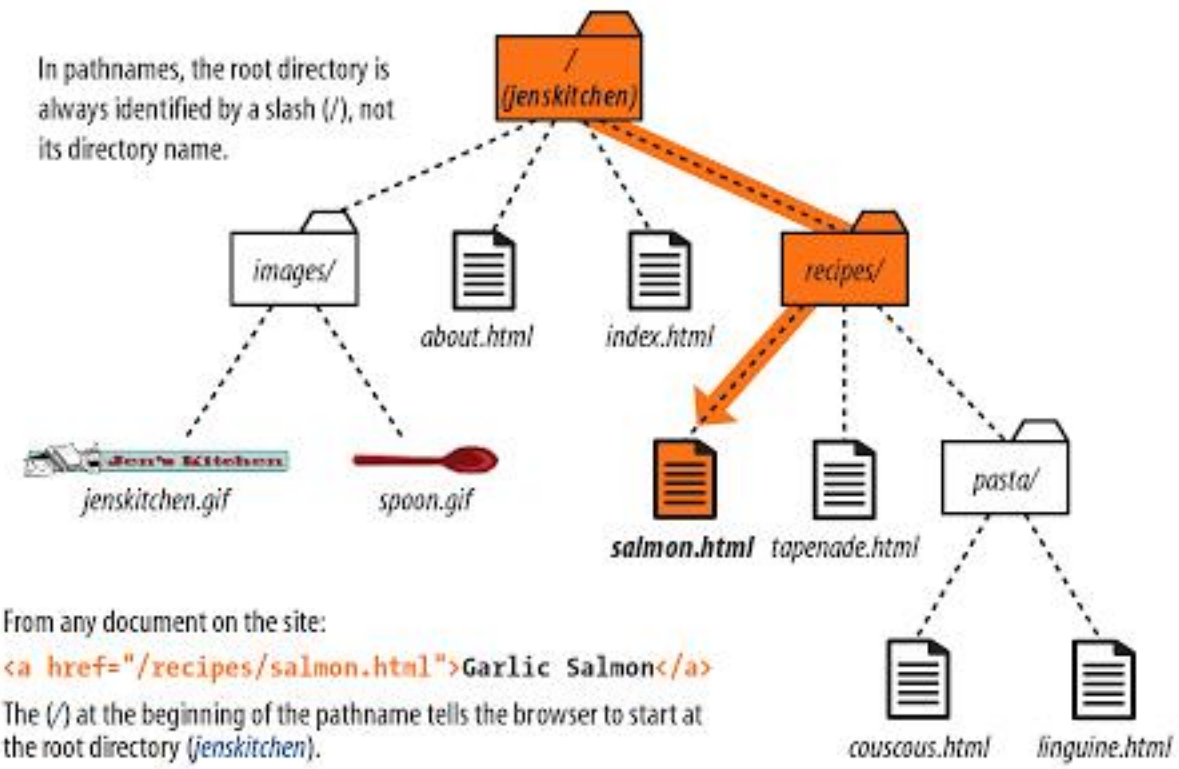


Figure 6-10. Writing a relative URL starting at the root directory.

Because this type of link starts at the root to describe the pathname, it will work from any document on the server, regardless of which subdirectory it may be located in. Site root relative links are useful for content that might not always be in the same directory, or for dynamically generated material. They also make it easy to copy and paste links between documents.

On the downside, however, the links won't work on your local machine, because they will be relative to your hard drive. You'll have to wait until the site is on the final server to check that links are working.

Obras protegidas por Direitos de Autor

It's the same for images

The **src** attribute in the **img** element works the same as the **href** attribute in anchors when it comes to specifying URLs. Since you'll most likely be using images from your own server, the **src** attributes within your image elements will be set to relative URLs.

Let's look at a few examples from the Jen's Kitchen site. First, to add an image to the *index.html* page, the markup would be:

```
<img src="images/jenskitchen.gif" alt="">
```

The URL says, "Look in the current directory (*jenskitchen*) for the *images* directory; in there you will find *jenskitchen.gif*."

Now for the *pièce de résistance*. Let's add an image to the file *couscous.html*:

```
<img src="../../images/spoon.gif" alt="">
```

This is a little more complicated than what we've seen so far. This pathname tells the browser to go up two directory levels to the top-level directory and, once there, look in the *images* directory for an image called *spoon.gif*. Whew!

Of course, you could simplify that path by going the site root relative route, in which case the pathname to *spoon.gif* (and any other file in the *images* directory) could be accessed like this:

```
<img src="/images/spoon.gif" alt="">
```

The trade-off is that you won't see the image in place until the site is uploaded to the server, but it does make maintenance easier once it's there.

A Little Help from Your Tools

If you use a WYSIWYG authoring tool to create your site, the tool generates relative URLs for you. Programs such as Adobe Dreamweaver and Microsoft Expression Web have built-in site management functions that adjust your relative URLs even if you reorganize the directory structure.

exercise 6-7 | Try a few more

Before we move on, you may want to try your hand at writing a few more relative URLs to make sure you've really gotten it. You can just write your answers on the page, or if you want to test your markup to see whether it works, make changes in the actual files. You'll need to add the text to the files to use as the link (for example, "Go to the Tapenade recipe" for the first question). Answers are in Appendix A.

1. Create a link on *salmon.html* to *tapenade.html*.

   `Go to the Tapenade recipe`

2. Create a link on *couscous.html* to *salmon.html*.

   `Try this with Garlic Salmon.`

3. Create a link on *tapenade.html* to *linguine.html*.

   `Try the Linguine with Clam Sauce`

4. Create a link on *linguine.html* to *about.html*.

   `About Jen's Kitchen`

5. Create a link on *tapenade.html* to *www.allrecipes.com*.

   `Go to Allrecipes.com`

NOTE

Most of the pathnames in Exercise 6-7 could be site root relative, but write them relative to the listed document for the practice.

Obras protegidas por Direitos de Autor

image not available

image not available

NOTE

Some developers help their brothers and sisters out by proactively adding `ids` *as anchors at the beginning of any thematic section of content (within a reasonable level, and depending on the site). That way other people can link back to any section in their content.*

Linking to a fragment in another document

You can link to a fragment in another document by adding the fragment name to the end of the URL (absolute or relative). For example, to make a link to the "H" heading of the glossary page from another document in that directory, the URL would look like this:

```
<a href="glossary.html#startH">See the Glossary, letter H</a>
```

You can even link to specific destinations in pages on other sites by putting the fragment identifier at the end of an absolute URL, like so:

```
<a href="http://www.example.com/glossary.html#startH">See the Glossary,
letter H</a>
```

Of course, you don't have any control over the named fragments in other people's web pages (see the note). The destination points must be inserted by the author of those documents in order for them to be available to you. The only way to know whether they are there and where they are is to "View Source" for the page and look for them in the markup. If the fragments in external documents move or go away, the page will still load; the browser will just go to the top of the page as it does for regular links.

exercise 6-8 | Linking to a fragment

Want some practice linking to specific destinations? Open the file ***glossary.html*** in the ***materials*** folder for this chapter. It looks just like the document in Figure 6-11.

1. Identify the `h2` "A" as a destination for a link by naming it "startA" with an `id` attribute.

   ```
   <h2 id="startA">A</h2>
   ```

2. Make the letter "A" at the top of the page a link to the identified fragment. Don't forget the #.

   ```
   <a href="#startA">A</a>
   ```

Repeat steps 1 and 2 for every letter across the top of the page until you really know what you are doing (or until you can't stand it anymore). You can help users get back to the top of the page, too.

3. Make the heading "Glossary" a destination named "top."

   ```
   <h1 id="top">Glossary</h1>
   ```

4. Add a paragraph element containing "TOP" at the end of each lettered section. Make "TOP" a link to the identifier that you just made at the top of the page.

   ```
   <p><a href="#top">TOP</a></
       p>
   ```

Copy and paste this code to the end of every letter section. Now your readers can get back to the top of the page easily throughout the document.

Targeting a New Browser Window

One problem with putting links on your page is that when people click on them, they may never come back. The traditional solution to this dilemma has been to make the linked page open in a new browser window. That way, your visitors can check out the link and still have your content available where they left it.

Before I provide the instructions for how to do it, I am going to strongly advise against it. First of all, tabbed browsers make it somewhat less likely that users will never find their way back to the original page. Furthermore, opening new windows is problematic for accessibility. New windows may be confusing to some users, particularly those who are accessing your site via a screen reader or even on a small-screen device. At the very least, new windows may be perceived as an annoyance rather than a convenience. Because it is common to configure your browser to block pop-up windows, you risk having the users miss out on the content in the new window altogether.

So consider carefully whether you need a separate window at all, and I'll tell you how in case you have a very good reason to do it. The method you use to open a link in a new browser window depends on whether you want to control its size. If the size of the window doesn't matter, you can use HTML markup alone. However, if you want to open the new window with particular pixel dimensions, then you need to use JavaScript.

Obras protegidas por Direitos de Autor

A new window with markup

To open a new window using markup, use the **target** attribute in the anchor (**a**) element to tell the browser the name of the window in which you want the linked document to open. Set the value of target to **_blank** or to any name of your choosing. Remember that with this method, you have no control over the size of the window, but it will generally open as a new tab or in a new window the same size as the most recently opened window in the user's browser.

Setting **target="_blank"** always causes the browser to open a fresh window. For example:

```
<a href="http://www.oreilly.com" target="_blank">O'Reilly</a>
```

If you target "_blank" for every link, every link will launch a new window, potentially leaving your user with a mess of open windows.

A better method is to give the target window a specific name, which can then be used by subsequent links. You can give the window any name you like ("new," "sample," whatever), as long as it doesn't start with an underscore. The following link will open a new window called "display":

```
<a href="http://www.oreilly.com" target="display">O'Reilly</a>
```

If you target the "display" window from every link on the page, each linked document will open in the same second window. Unfortunately, if that second window stays hidden behind the user's current window, it may look as though the link simply didn't work.

Pop-up windows

It is possible to open a window with specific dimensions and various parts of the browser chrome (toolbars, scrollbars, etc.) turned on or off; however, it takes JavaScript to do it. These are known as pop-up windows, and they are commonly used for advertising. In fact, they've become such a nuisance that many browsers have preferences for turning them off completely. Furthermore, in a world where sites are accessed on small, mobile devices, popping up windows at specific pixel dimensions has no place.

That said, if you have a valid reason to open a new browser window at a specific size, I recommend this tutorial article by Peter-Paul Koch at Quirksmode: *www.quirksmode.org/js/popup.html*.

Mail Links

Here's a nifty little linking trick: the **mailto** link. By using the **mailto** protocol in a link, you can link to an email address. When the user clicks on a **mailto** link, the browser opens a new mail message preaddressed to that address in a designated mail program.

Obras protegidas por Direitos de Autor

A sample **mailto** link is shown here:

```
<a href="mailto:alklecker@example.com">Contact Al Klecker</a>
```

As you can see, it's a standard anchor element with the **href** attribute. But the value is set to **mailto:***name@address.com*.

The browser has to be configured to launch a mail program, so the effect won't work for 100% of your audience. If you use the email address itself as the linked text, nobody will be left out if the **mailto** function does not work (a nice little example of progressive enhancement).

Spam-Bots

Be aware that by putting an email address in your document source, you will make it susceptible to receiving unsolicited junk email (known as spam). People who generate spam lists sometimes use automated search programs (called bots) to scour the Web for email addresses.

If you want your email to display on the page in a way that humans can figure it out but robots can't, you can deconstruct the address in a way that is still understandable to people, for example, "jen [-at-] oreilly [dot] com."

That trick won't work in a **mailto** link, because the accurate email address must be provided as an attribute value. One solution is to encrypt the email address using JavaScript. The Enkoder Form at Hivelogic (*hivelogic.com/enkoder/*) does this for you. Simply enter the link text and the email address, and Enkoder generates code that you can copy and paste into your document.

Otherwise, if you don't want to risk getting spammed, keep your email address out of your HTML document.

Telephone Links

Keep in mind that the smartphones people are using to access your website can also be used to make phone calls! Why not save your visitors a step by letting them dial a phone number on your site simply by tapping on it on the page? The syntax uses the **tel:** scheme and is very simple.

```
<a href="tel:+18005551212">Call us free at (800) 555-1212</a>
```

When mobile users tap the link, they get an alert box asking them to confirm that they'd like to call the number. This feature is supported on most mobile devices, including iOS, Android, BlackBerry, Symbian, Internet Explorer, and Opera Mini. The iPad and iPod Touch can't make a call, but they will offer to create a new contact from the number. Nothing happens when desktop users click the link. If that bothers you, you could use a CSS rule that hides the link for non-mobile devices (unfortunately, that is beyond the scope of this discussion).

There are a few best practices for using telephone links:

- It is recommended that you include the full international dialing number, including the country code, for the **tel:** value because there is no way of knowing where the user will be accessing your site.
- Also include the telephone number in the content of the link so that if the link doesn't work, the telephone number is still available.
- Android and iPhone have a feature that detects phone numbers and automatically turns them into links. Unfortunately, some 10-digit numbers that are not telephone numbers might get turned into links, too. If your document has strings of numbers that might get confused as phone numbers, you can turn auto-detection off by including the following **meta** element in the **head** of your document.

  ```
  <meta name="format-detection" content="telephone=no">
  ```

  For BlackBerry devices, use the following:

  ```
  <meta http-equiv="x-rim-auto-match" content="none">
  ```


Obras protegidas por Direitos de Autor

Test Yourself

The most important lesson in this chapter is how to write URLs for links and images. Here's another chance to brush up on your pathname skills.

Using the directory hierarchy shown in Figure 6-12, write out the markup for the following links and graphics. I filled in the first one for you as an example. The answers are located in Appendix A.

This diagram should provide you with enough information to answer the questions. If you need hands-on work to figure them out, the directory structure is available in the *test* directory in the materials for this chapter. The documents are just dummy files and contain no content.

TIP

The `../` (or multiples of them) always appears at the beginning of the pathname and never in the middle. If the pathnames you write have `../` in the middle, you've done something wrong.

1. In *index.html* (the site's home page), write the markup for a link to *tutorial.html*.

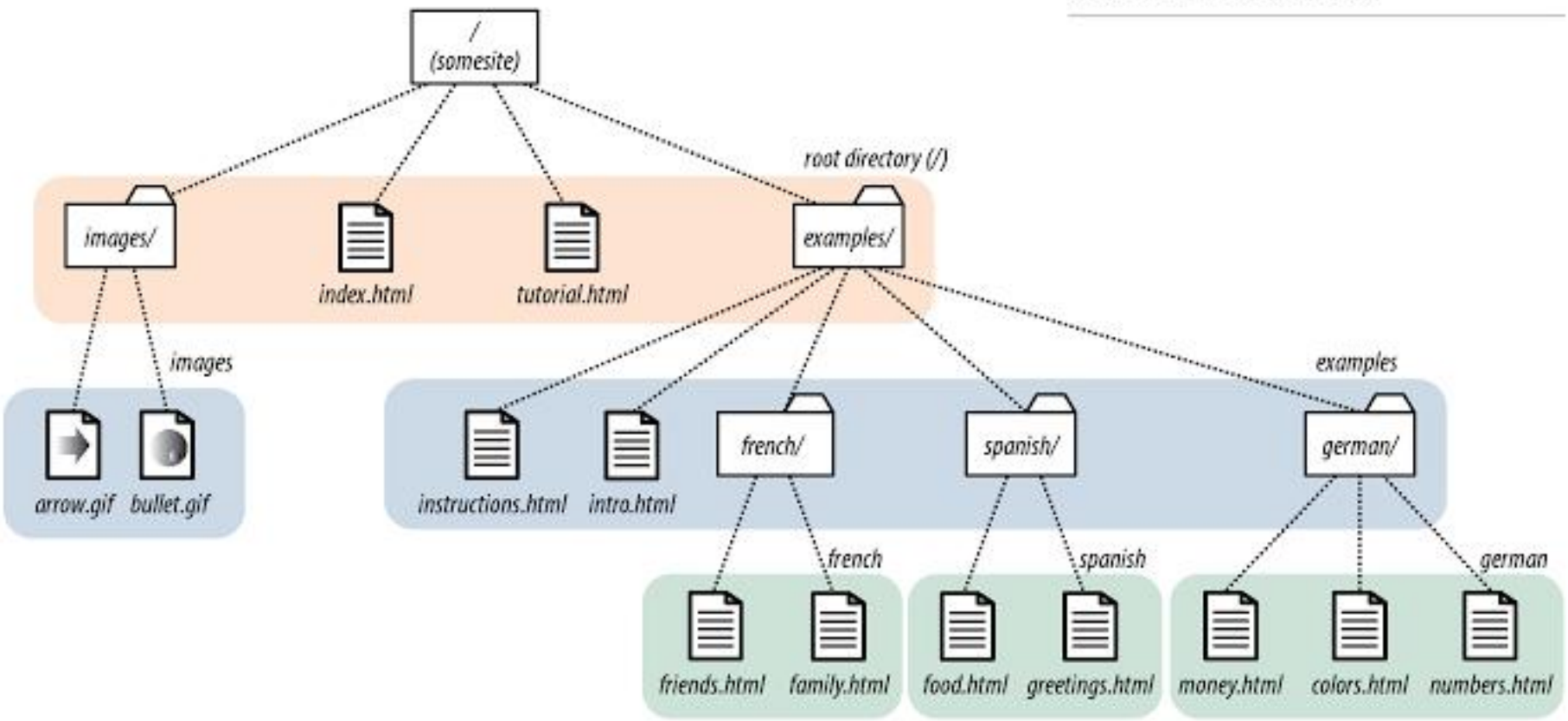
```
<a href="tutorial.html">...</a>
```

2. In *index.html*, write the anchor element for a link to *instructions.html*.

3. Create a link to *family.html* from the page *tutorial.html*.

4. Create a link to *numbers.html* from the *family.html* page, but this time, start with the root directory.

Figure 6-12. The directory structure for the Test Yourself questions.



Obras protegidas por Direitos de Autor

5. Create a link back to the home page (*index.html*) from the page *instructions.html*.

6. In the file *intro.html*, create a link to the website for this book (*www.learningwebdesign.com*).

7. Create a link to *instructions.html* from the page *greetings.html*.

8. Create a link back to the home page (*index.html*) from *money.html*.

We haven't covered the image (**img**) element in detail yet, but you should be able to fill in the relative URLs after the src attribute to specify the location of the image files for these examples.

9. To place the graphic *arrow.gif* on the page *index.html*, use this URL:

```
<img src="                                    " alt="">
```

10. To place the graphic *arrow.gif* on the page *intro.html*, use this URL:

```
<img src="                                    " alt="">
```

11. To place the graphic *bullet.gif* on the *friends.html* page, use this URL:

```
<img src="                                    " alt="">
```

Element Review: Links

There's really only one element relevant to creating hypertext links:

Element and attributes	Description
a	Anchor (hypertext link) element
href="url"	Location of the target file

Obras protegidas por Direitos de Autor

ADDING IMAGES

IN THIS CHAPTER

Adding images to a web page

Using the `src`, `alt`, `width`, and `height` attributes

A web page with all text and no pictures isn't much fun. The Web's explosion into mass popularity was due in part to the fact that there were images on the page. Before images, the Internet was a text-only tundra.

Images appear on web pages in two ways: embedded in the inline content or as background images. Background images are added using Cascading Style Sheets and are talked about at length in Chapter 13, Colors and Backgrounds. With the emergence of standards-driven design and its mission to keep all matters of presentation out of the document structure, there has been a shift away from using inline images for purely decorative purposes. See the sidebar "Images Move to the Background" on the following page for more information on this trend.

In this chapter, we'll focus on embedding image content into the document using the **`img`** element. Use the **`img`** element when the image is the content, such as product shots, gallery images, ads, illustrations, and so on...I think you get the idea.

First, a Word on Image Formats

We'll get to the **`img`** element and markup examples in a moment, but first it's important to know that you can't put just any image on a web page. In order to be displayed inline, images must be in the GIF, JPEG, or PNG file format. Chapter 21, Web Graphics Basics, explains these formats and the image types they handle best. In addition to being in an appropriate format, image files need to be named with the proper suffixes—*.gif*, *.jpg* (or *.jpeg*), and *.png*, respectively—in order to be recognized by the browser.

If you have a source image that is in another popular format, such as TIFF, BMP, or EPS, you'll need to convert it to a web format before you can add it to the page. If, for some reason, you must keep your graphic file in its original format (for example, a file for a CAD program or an image in a vector format), you can make it available as an external image by making a link directly to the image file, like this:

```
<a href="architecture.eps">Get the drawing</a>
```


Obras protegidas por Direitos de Autor

Images Move to the Background

Images that are used purely for decoration have more to do with presentation than document structure and content. For that reason, they should be controlled with a style sheet rather than the markup.

Using CSS, it is possible to place an image in the background of the page or in any text element (a **div**, **h1**, **li**… you name it). These techniques are introduced in Chapter 13, Colors and Backgrounds.

There are several benefits to specifying decorative images only in an external style sheet and keeping them out of the document structure. Not only does it make the document cleaner and more accessible, but it also makes it easier to make changes to the look and feel of a site than when presentational elements are interspersed in the content.

For inspiration on how visually rich a web page can be with no **img** elements at all, look at the examples in the "Select a Design" section of the CSS Zen Garden site at *www.csszengarden.com*.

Browsers use helper applications to display media they can't handle alone. The browser matches the suffix of the file in the link to the appropriate helper application. The external image may open in a separate application window or within the browser window if the helper application is a plug-in, such as the QuickTime plug-in. The browser may also ask the user to save the file or open an application manually. It is also possible that it won't be able to be opened at all.

Without further ado, let's take a look at the **img** element and its required and recommended attributes.

The img Element

`<img>`

Adds an inline image

The **img** element tells the browser, "Place an image here." You've already gotten a glimpse of it used to place banner graphics in the examples in Chapters 4 and 5. You can also place an image element right in the flow of the text at the point where you want the image to appear, as in the following example. Images stay in the flow of text and do not cause any line breaks (HTML5 calls this a phrasing element), as shown in Figure 7-1.

```
<p>I had been wanting to go to Tuscany <img src="tuscany.jpg" alt="">
 for a long time, and I was not disappointed.</p>
```

Figure 7-1. By default, images are aligned with the baseline of the surrounding text, and they do not cause a line break.

When the browser sees the **img** element, it makes a request to the server and retrieves the image file before displaying it on the page. On a fast network with a fast computer, even though a separate request is made for each image file, the page usually appears to arrive instantaneously. On mobile devices with slow network connections, we may be well aware of the wait for images to be fetched one at a time. The same is true for users still using dial-up Internet connections or other slow networks, like the expensive WiFi at luxury hotels.

When designing mobile web experiences, it is wise to limit the number of server requests in general, which means carefully considering the number of images on the page.

Obras protegidas por Direitos de Autor

The **src** and **alt** attributes shown in the sample are required. The **src** attribute tells the browser the location of the image file. The **alt** attribute provides alternative text that displays if the image is not available. We'll talk about **src** and **alt** a little more in upcoming sections.

The src and alt attributes are required in the img element.

There are a few other things of note about the **img** element:

- It is an empty element, which means it doesn't have any content. You just place it in the flow of text where the image should go.
- If you choose to write in the stricter XHTML syntax, you need to terminate (close) the empty **img** element with a slash like so: **<img />**.
- It is an inline element, so it behaves like any other inline element in the text flow. Figure 7-2 demonstrates the inline nature of image elements. When the browser window is resized, the line of images reflows to fill the new width.
- The **img** element is what's known as a replaced element because it is replaced by an external file when the page is displayed. This makes it different from text elements that have their content right there in the source (and thus are non-replaced).
- By default, the bottom edge of an image aligns with the baseline of text, as shown in Figures 7-1 and 7-2. Using CSS, you can float the image to the right or left margin and allow text to flow around it, control the space and borders around the image, and change its vertical alignment. We'll talk about those styles in Part III.

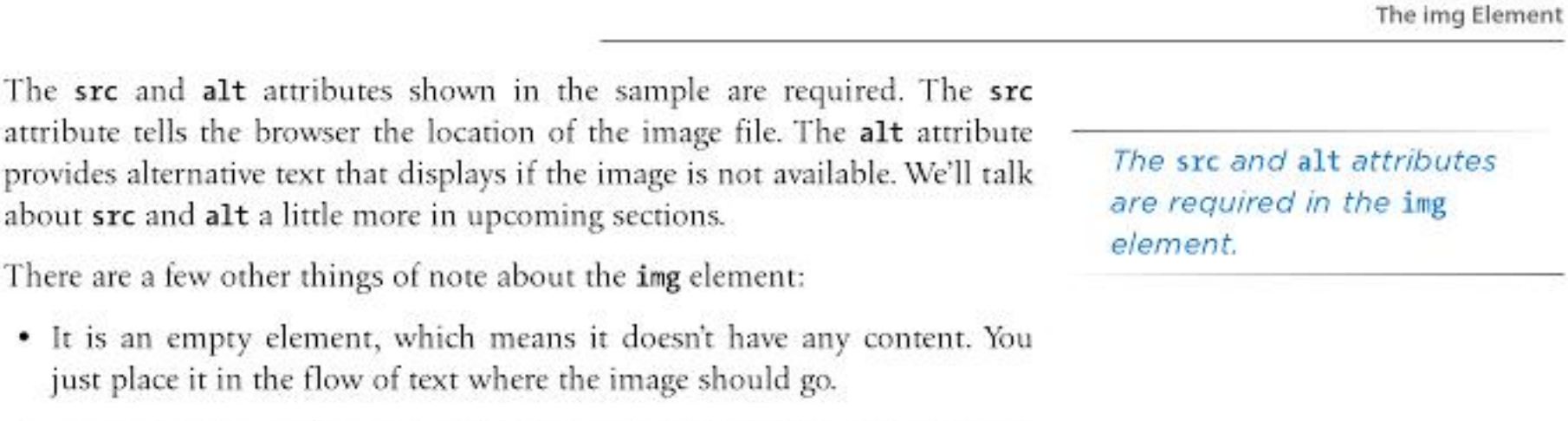


Figure 7-2. Inline images are part of the normal document flow. They reflow when the browser window is resized.

Providing the location with src

`src="URL"`

Source (location) of the image

The value of the **src** attribute is the URL of the image file. In most cases, the images you use on your pages will reside on your server, so you will use

Obras protegidas por Direitos de Autor

relative URLs to point to them. If you just read Chapter 6, Adding Links, you should be pretty handy with writing relative URLs by now. In short, if the image is in the same directory as the HTML document, you can just refer to the image by name in the **src** attribute:

```
<img src="icon.gif" alt="">
```

Developers usually organize the images for a site into a directory called *images*, *assets*, or *graphics*. There may even be separate image directories for each section of the site. If an image is not in the same directory as the document, you need to provide the pathname to the image file.

```
<img src="/images/arrow.gif" alt="">
```

Of course, you can place images from other websites as well (just be sure that you have permission to do so). Just use an absolute URL, like this:

```
<img src="http://www.example.com/images/smile.gif" alt="">
```

TIP

Take Advantage of Caching

Here's a tip for making images display more quickly and reducing the traffic to your server. If you use the same image in multiple places on your site, be sure each **img** element is pointing to the same image file on the server.

When a browser downloads an image file, it stores it in the disk cache (a space for temporarily storing files on the hard disk). That way, if it needs to redisplay the page, it can just pull up a local copy of the source document and image files without making a new trip out to the remote server.

When you use the same image repetitively in a page or a site, the browser only needs to download the image once. Every subsequent instance of the image is grabbed from the local cache, which means less traffic for the server and faster display for the end user.

The browser recognizes an image by its entire pathname, not just the filename, so if you want to take advantage of file caching, be sure that each instance of your image is pointing to the same image file on the server, not multiple copies of the same image file in different directories.

Providing alternate text with alt

`alt="text"`

Alternative text

Every **img** element must also contain an **alt** attribute that is used to provide a brief description of the image for those who are not able to see it, such as users with screen readers, Braille, or even small mobile devices. Alternate text (also referred to as alt text) should serve as a substitute for the image content—serving the same purpose and presenting the same information.

```
<p>If you're <img src="happyface.gif" alt="happy"> and you know it clap
   your hands.</p>
```

A screen reader might indicate the image by reading its **alt** value this way:

> *"If you're image happy and you know it clap your hands."*

If an image does not add anything meaningful to the text content of the page, it is recommended that you leave the value of the **alt** attribute empty, as shown in the following example and other examples in this chapter (you may also consider whether it is more appropriately handled as a background image with CSS, but I digress). Note that there is no character space between the quotation marks.

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

Do not omit the **alt** attribute altogether, however, because it will cause the document to be invalid (validating documents is covered in Chapter 3, Some Big Concepts You Need to Know). For each inline image on your page, consider what the alternative text would sound like when read aloud and whether that enhances or is just obtrusive to a screen-reader user's experience.

Alternative text may benefit users with graphical browsers as well. If a user has opted to turn images off in the browser preferences or if the image simply fails to load, the browser may display the alternative text to give the user

Obras protegidas por Direitos de Autor

image not available

TIP

Using a Browser to Find Pixel Dimensions

You can find the pixel dimensions of an image by opening it in an image-editing program, of course, but did you know you can also use a web browser?

Using Chrome, Firefox, or Safari (but, sorry, not Internet Explorer), simply open the image file, and its pixel dimensions display in the browser's title bar along with the filename. It's a handy shortcut I use all the time because I always seem to have a browser running.

Providing width and height dimensions

`width="number"`

Image width in pixels

`height="number"`

Image height in pixels

The **width** and **height** attributes indicate the dimensions of the image in number of pixels. Sounds mundane, but these attributes can speed up the time it takes to display the final page by seconds. Browsers use the specified dimensions to hold the right amount of space in the layout while the images are loading rather than reconstructing the page each time a new image arrives.

And that's great if you are designing one version of your page with one fixed image size. However, in these days of responsive web design, it is common to create several versions of the same image and send a small one to small mobile devices and a larger image for large-screen devices (and rescale the images to fit for sizes in between). If you are scaling images in a responsive design or delivering multiple image sizes, do not use **width** and **height** attributes in the markup.

With this caveat in mind, let's look at how **width** and **height** work for those cases when it is appropriate to use them.

Match values with actual pixel size

Be sure that the pixel dimensions you specify are the actual dimensions of the image. If the pixel values differ from the actual dimensions of your image, the browser resizes the image to match the specified values (Figure 7-4).

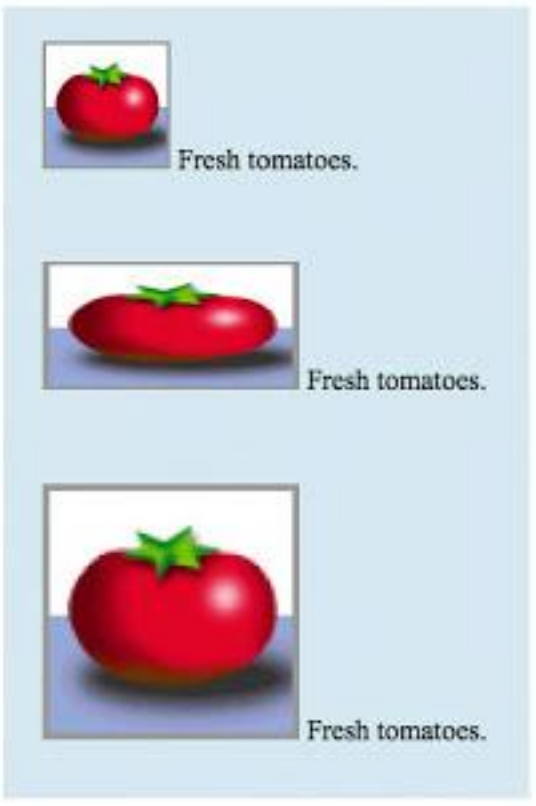


Figure 7-4. Browsers resize images to match the provided `width` *and* `height` *values. It is strongly recommended not to resize images in this way.*

Although it may be tempting to resize images in this manner, you should avoid doing so. Even though the image may appear small on the page, the large image with its corresponding large file size still needs to download. It is better to resize the image in an image-editing program and then place it at actual size on the page. Not only that, but resizing with attributes usually results in a blurry and deformed image. In fact, if your images ever look fuzzy when viewed in a browser, the first thing to check is that the **width** and **height** values match the dimensions of the image exactly.

Obras protegidas por Direitos de Autor

exercise 7-1 | Adding and linking images

You're back from Italy and it's time to post about some of your travels. In this exercise, you'll add thumbnail images to a travelog and make them link to pages with full-sized versions.

All the thumbnails and photos you need have been created for you, and I've given you a head start on the HTML files as well. Everything is available at *www.learningwebdesign.com/4e/materials*. Put a copy of the ***tuscany*** folder on your hard drive, making sure to keep it organized as you find it. As always, the resulting markup is listed in Appendix A.

This little site is made up of a main page (***index.html***) and three separate HTML documents containing each of the larger image views (Figure 7-5). First, we'll add the thumbnails, and then we'll add the full-size versions to their respective pages. Finally, we'll make the thumbnails link to those pages. Let's get started. Open the file ***index.html***, and add the small thumbnail images to this page to accompany the text. I've done the first one for you:

```
<h2>Pozzarello</h2>
<p><img src="thumbnails/window_thumb.jpg"
alt="view from bedroom window" width="75"
height="100"> The house we stayed in was called
Pozzarello...
```

I've put the image at the beginning of the paragraph, just after the opening **<p>** tag. Because all of the thumbnail images are located in the *thumbnails* directory, I provided the pathname in the URL. I also added a description of the image and the width and height dimensions in pixels (px).

Now it's your turn. Add the image ***countryside_thumb.jpg*** (100px wide × 75px tall) and ***sienna_thumb.jpg*** (75 × 100) at the beginning of the paragraphs in their respective sections. Be sure to include the pathname, an alternative text description, and pixel dimensions.

When you are done, save the file and then open it in the browser to be sure that the images are visible and appear at the right size.

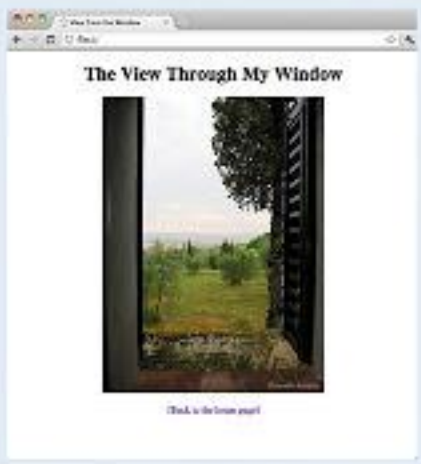


Figure 7-5. Travel photo site.

Obras protegidas por Direitos de Autor

1. Next, add the images to the individual HTML documents. I've done ***window.html*** for you:

   ```
   <h1>The View Through My Window</h1>
   <p><img src="photos/window.jpg" alt="view out the
   window of the rolling Tuscan hills" width="375"
   height="500"></p>
   ```

   Notice that the full-size images are in a directory called ***photos***, so that needs to be reflected in the pathnames.

   Add images to ***countryside.html*** and ***sienna.html***, following my example. Hint: all of the images are 500 pixels on their widest side and 375 pixels on their shortest side, although the orientation varies.

   Save each file, and check your work by opening them in the browser window.

2. Back in ***index.html***, link the thumbnails to their respective files. I've done the first one here.

   ```
   <h2>Pozzarello</h2>
   <p><a href="window.html"><img src="thumbnails/
   window_thumb.jpg" alt="view from the bedroom
   window" width="75" height="100"></a></p>
   ```

   Notice that the URL is relative to the current document (***index.html***), not to the location of the image (the ***thumbnails*** directory).

   Make the remaining thumbnail images links to each of the documents. If all the images are visible and you are able to link to each page and back to the home page again, then congratulations, you're done!

Like a little more practice?

If you'd like more practice, you'll find three additional images (***sweets.jpg***, ***cathedral.jpg***, and ***lavender.jpg***) with their thumbnail versions (***sweets_thumb.jpg***, ***cathedral_thumb.jpg***, and ***lavender_thumb.jpg***) in their appropriate directories. This time, you'll need to add your own descriptions to the home page and create the HTML documents for the full-size images from scratch.

For an added challenge, create a new directory called ***photopages*** in the ***tuscany*** directory. Move ***countryside.html*** and ***sienna.html*** into that directory, and then update the URLs on those pages so that the images are visible again.

A Window in a Window

As long as we're talking about embedding things on a page, I thought I'd tell you about the `iframe` element that lets you embed a separate HTML document or other resource in a document. What you see on the page is a floating or inline "frame" that displays the document with its own set of scrollbars if the embedded document is too long to fit (Figure 7-6).

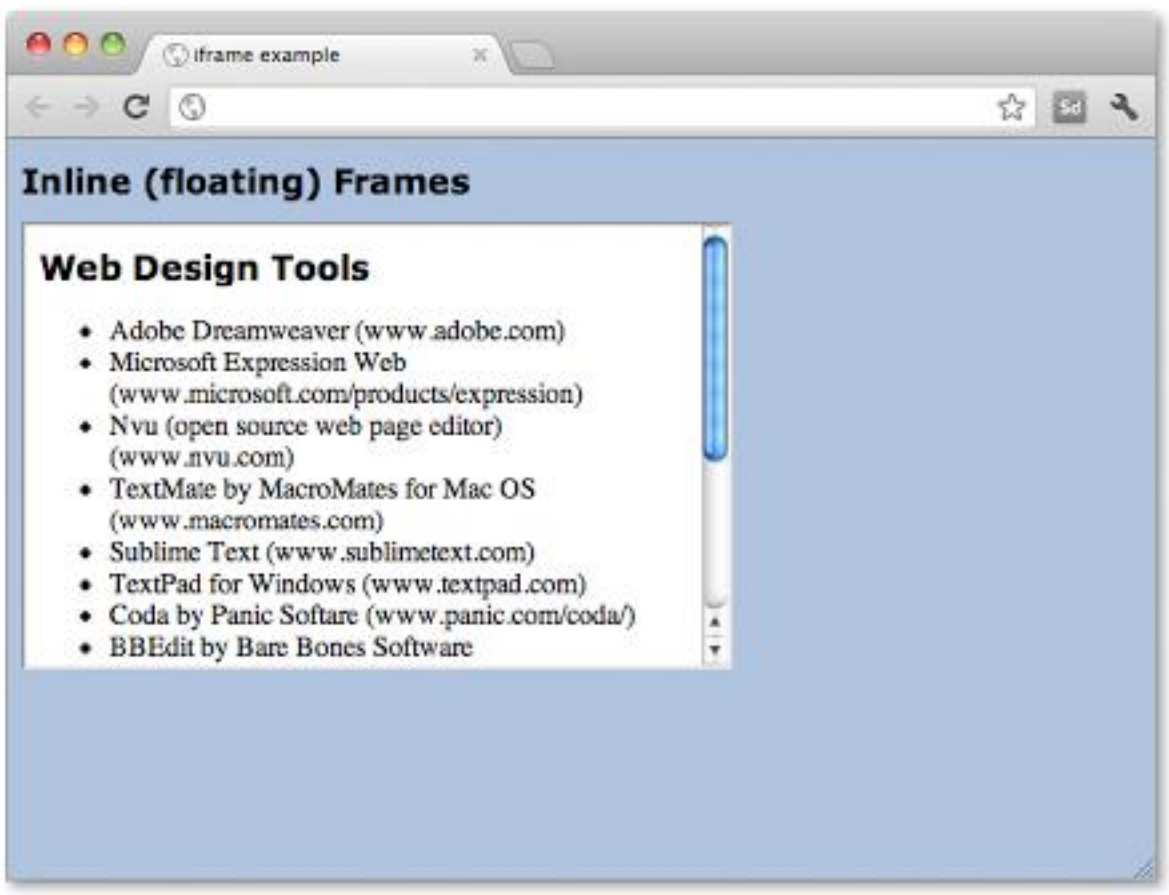


Figure 7-6. Inline frames (added with the `iframe` element) are like a browser window within the browser that displays external HTML documents and resources.

You place an inline frame on a page similarly to an image, specifying the source (`src`) of its content as well as its width and height. The content in the `iframe` element itself displays on browsers that don't support the element. This example displays a document called *list.html* in an inline frame.

Obras protegidas por Direitos de Autor

```
<h1>Inline (floating) Frames</h1>
<iframe src="list.html" width="400" height="250">
Your browser does not support inline frames. Read the <a href="list.
    html">list</a>.
</iframe>
```

You don't see inline frames much in the wild, but developers sometimes use them to keep third-party content such as interactive ads or other widgets quarantined so they don't interfere with the scripting and contents of the rest of the page.

Test Yourself

Images are a big part of the web experience. Answer these questions to see how well you've absorbed the key concepts of this chapter. The correct answers can be found in Appendix A.

1. Which attributes must be included in every `img` element?

2. Write the markup for adding an image called *furry.jpg* that is in the same directory as the current document.

3. Why is it necessary to include alternative text? Name two reasons.

4. What is the advantage of including `width` and `height` attributes for every graphic on the page? When should you leave them out?

5. What might be going wrong if your images don't appear when you view the page in a browser? There are three possible explanations.

Obras protegidas por Direitos de Autor

Element Review: Images

We covered just one element in this chapter:

Element and attributes	Description
`img`	Inserts an inline image.
`src="url"`	The location of the image file.
`alt="text"`	Alternative text.
`width="number"`	Width of the graphic.
`height="number"`	Height of the graphic.
`usemap="usemap"`	Indicates a client-side image map.
`title="text"`	Provides a "tooltip" when the user mouses over the image. Can be used for supplemental information about the image.
`iframe`	Inserts an inline browsing context (window).
`height="number"`	Height of the frame in pixels.
`src="url"`	Resource of the display in the frame.
`width="number"`	Width of the frame in pixels.

Obras protegidas por Direitos de Autor

CHAPTER 8

TABLE MARKUP

Before we launch into the markup for tables, let's check in with our progress so far. We've covered a lot of territory: how to establish the basic structure of an HTML document, how to mark up text to give it meaning and structure, how to make links, and how to embed images on the page.

This chapter and the next, Chapter 9, Forms, describe the markup for specialized content that you might not have a need for right away. If you're getting antsy to make your pages look good, skip right to Part III and start playing with Cascading Style Sheets. The tables and forms chapters will be here when you're ready for them.

Are you still with me? Great. Let's talk tables. We'll start out by reviewing how tables should be used, then learn the elements used to create them with markup. Remember, this is an HTML chapter, so we're going to focus on the markup that structures the content into tables, and we won't be concerned with how the tables look. Like any web content, the appearance (or presentation, as we say in the web dev biz) of tables should be handled with style sheets, which you'll learn about in Chapter 18, CSS Techniques.

IN THIS CHAPTER

How tables are used
Basic table structure
The importance of headers
Spanning rows and columns
Cell padding and spacing
Making tables accessible

How Tables Are Used

HTML tables were created for instances when you need to add tabular material (data arranged into rows and columns) to a web page. Tables may be used to organize calendars, schedules, statistics, or other types of information, as shown in Figure 8-1. Note that "data" doesn't necessarily mean numbers. A table cell may contain any sort of information, including numbers, text elements, and even images and multimedia objects.

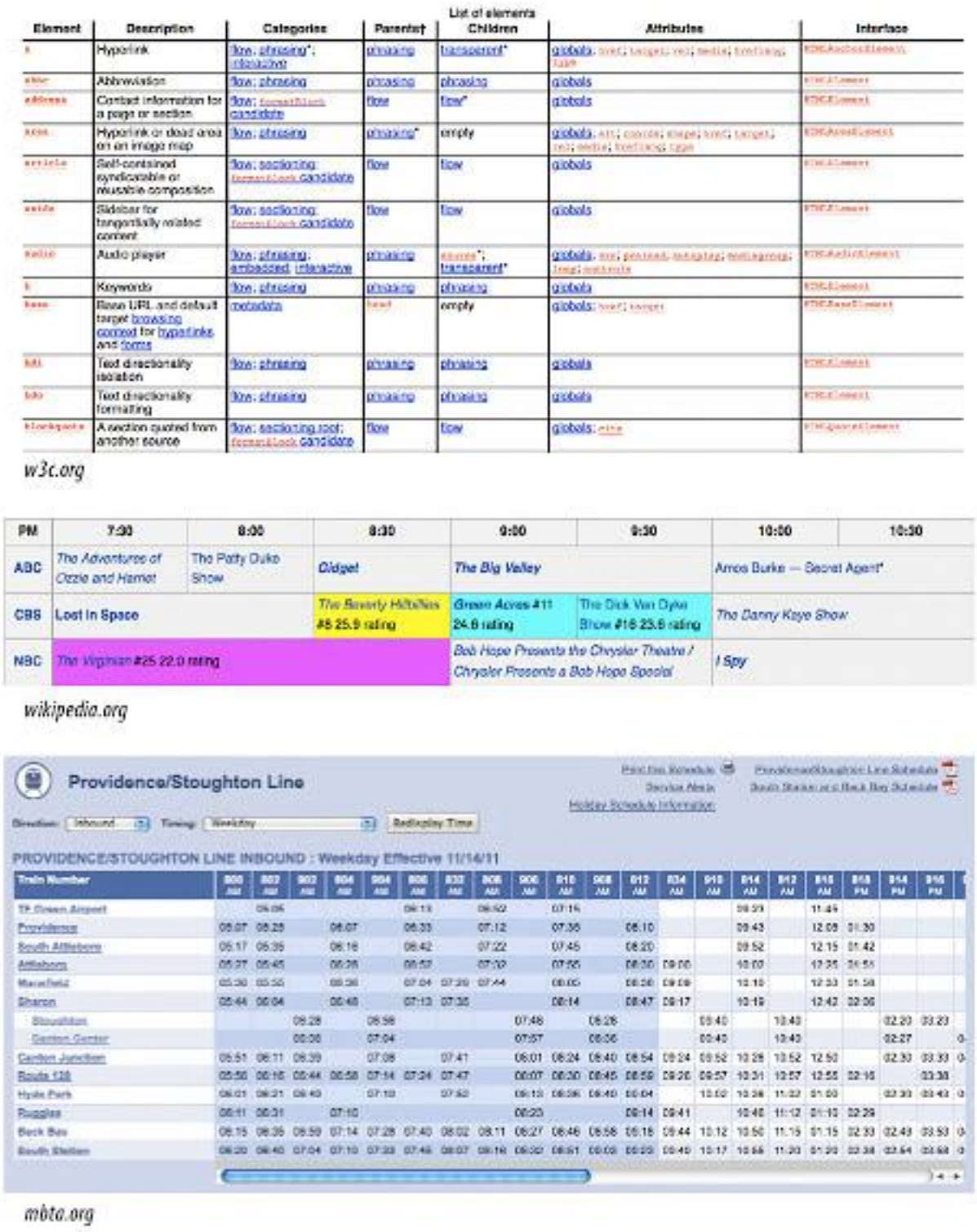
Obras protegidas por Direitos de Autor

The Trouble with Tables

Large tables, such as those shown in Figure 8-1, can be difficult to use on small-screen devices. By default, they are shrunk to fit the screen width, rendering the text in the cells too small to be read. Users can zoom in to read the cells, but then only a few cells may be visible at a time and it is difficult to parse the organization of headings and columns.

To be honest, as of this writing, we are just starting to figure out how best to handle tabular material on small screens. One approach is to replace the table with a graphic representation, such as a pie chart, on mobile devices. Of course, this will work only for certain types of tables. For simple two- or three-column tables, consider using a `dl` list to represent the information instead for more flexibility. Another approach is to put an indication of the table (such as an image of the top of it) that links to a separate screen with the full table for those who are interested. Chris Coyier proposes a clever solution in his article "Responsive Data Tables" (*css-tricks.com/9096-responsive-data-tables/*) that describes how to use CSS to reformat the table as a long, narrow list that fits better in a smartphone screen. See also the clever solution proposed by Filament Group (think of them as the Super Friends of responsive design) at *filamentgroup.com/lab/responsive_design_approach_for_complex_multicolumn_data_tables/*.

There may be new solutions by the time you read this, but it is important to always keep the mobile, small-screen experience in mind as you design any web content.



Element	Description	Categories	Parents†	List of elements Children	Attributes	Interface
a	Hyperlink	flow; phrasing*; interactive	phrasing	transparent*	globals, href, target, rel, media, hreflang, type	HTMLAnchorElement
abbr	Abbreviation	flow; phrasing	phrasing	phrasing	globals	HTMLElement
address	Contact information for a page or section	flow; formattingBlock candidate	flow	flow*	globals	HTMLElement
area	Hyperlink or dead area on an image map	flow; phrasing	phrasing*	empty	globals, alt, coords, shape, href, target, rel, media, hreflang, type	HTMLAreaElement
article	Self-contained syndicatable or reusable composition	flow; sectioning; formattingBlock candidate	flow	flow	globals	HTMLElement
aside	Sidebar for tangentially related content	flow; sectioning; formattingBlock candidate	flow	flow	globals	HTMLElement
audio	Audio player	flow; phrasing; embedded; interactive	phrasing	source*; transparent*	globals, src, preload, autoplay, mediagroup, loop, controls	HTMLAudioElement
b	Keywords	flow; phrasing	phrasing	phrasing	globals	HTMLElement
base	Base URL and default target browsing context for hyperlinks and forms	metadata	head	empty	globals, href, target	HTMLBaseElement
bdi	Text directionality isolation	flow; phrasing	phrasing	phrasing	globals	HTMLElement
bdo	Text directionality formatting	flow; phrasing	phrasing	phrasing	globals	HTMLElement
blockquote	A section quoted from another source	flow; sectioning root; formattingBlock candidate	flow	flow	globals, cite	HTMLQuoteElement

w3c.org

PM	7:00	8:00	8:30	9:00	9:30	10:00	10:30
ABC	*The Adventures of Ozzie and Harriet*	The Patty Duke Show	*Gidget*	*The Big Valley*		Amos Burke — Secret Agent*	
CBS	Lost in Space		*The Beverly Hillbillies* #5 25.9 rating	Green Acres #11 24.6 rating	The Dick Van Dyke Show #16 23.6 rating	*The Danny Kaye Show*	
NBC	*The Virginian* #25 22.0 rating			*Bob Hope Presents the Chrysler Theatre / Chrysler Presents a Bob Hope Special*		*I Spy*	

wikipedia.org

mbta.org

Figure 8-1. Examples of tables used for tabular information, such as charts, calendars, and schedules.

In visual browsers, the arrangement of data in rows and columns gives readers an instant understanding of the relationships between data cells and their respective header labels. Bear in mind when you are creating tables, however, that some readers will be hearing your data read aloud with a screen reader or reading Braille output. Later in this chapter, we'll discuss measures you can take to make table content accessible to users who don't have the benefit of visual presentation.

In the days before style sheets, tables were the only option for creating multicolumn layouts or controlling alignment and whitespace. Layout tables, particularly the complex nested table arrangements that were once standard web design fare, have gone the way of the dodo. This chapter focuses on HTML tables as they are intended to be used.

Obras protegidas por Direitos de Autor

Minimal Table Structure

Let's take a look at a simple table to see what it's made of. Here is a small table with three rows and three columns that lists nutritional information.

Menu item	Calories	Fat (g)
Chicken noodle soup	120	2
Caesar salad	400	26

`<table>...</table>`

Tabular content (rows and columns)

`<tr>...</tr>`

Table row

`<th>...</th>`

Table header

`<td>...</td>`

Table cell data

Figure 8-2 reveals the structure of this table according to the HTML table model. All of the table's content goes into cells that are arranged into rows. Cells contain either header information (titles for the columns, such as "Calories") or data, which may be any sort of content.

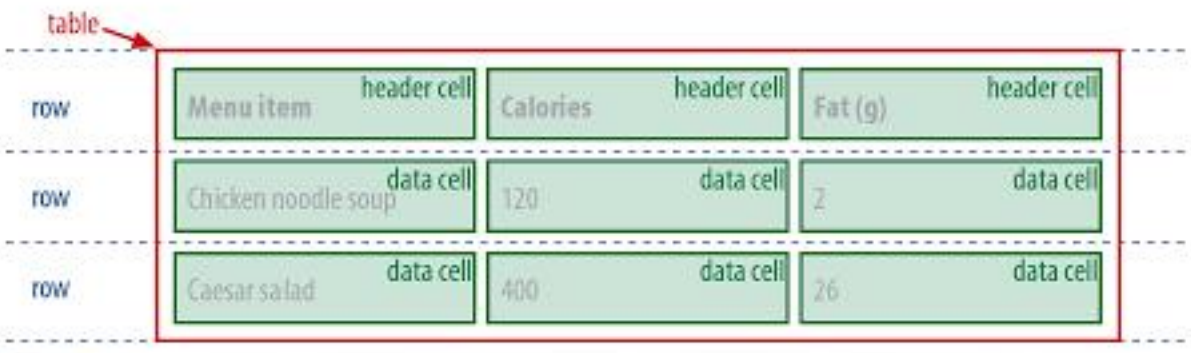


Figure 8-2. Tables are made up of rows that contain cells. Cells are the containers for content.

Simple enough, right? Now let's look at how those parts translate into elements (Figure 8-3).

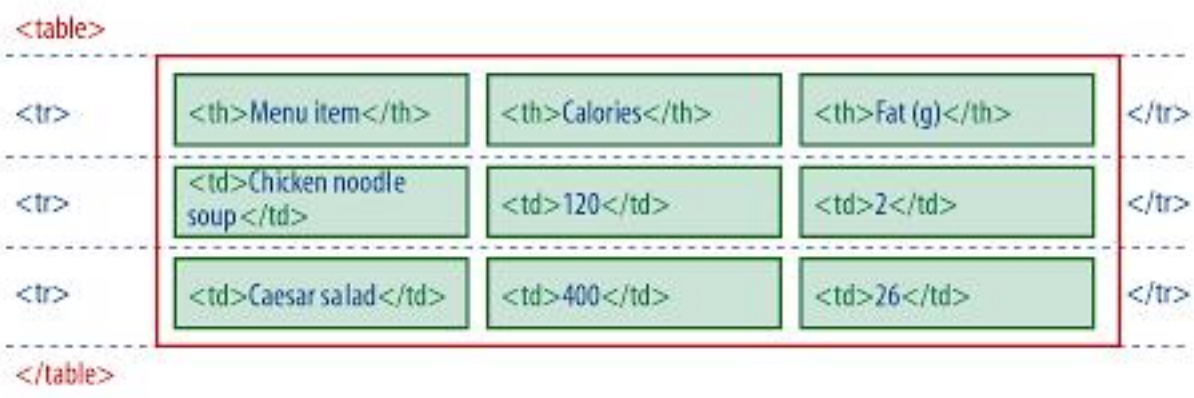


Figure 8-3. The elements that make up the basic structure of a table.

Figure 8-3 shows the elements that identify the table (**table**), rows (**tr**, for "table row"), and cells (**th**, for "table headers," and **td**, for "table data"). Cells are the heart of the table, because that's where the actual content goes. The other elements just hold things together.

Obras protegidas por Direitos de Autor

What we don't see are column elements (see note). The number of columns in a table is determined by the number of cells in each row. This is one of the things that make HTML tables potentially tricky. Rows are easy—if you want the table to have three rows, just use three **tr** elements. Columns are different. For a table with four columns, you need to make sure that every row has four **td** or **th** elements; the columns are implied.

NOTE

There are two column-related elements in HTML5: **col** *for identifying a column and* **colgroup** *for establishing related groups of columns. They were created to add a layer of information about the table that can potentially speed up its display, but they are not part of HTML's row-centric table model. See the sidebar "Advanced Table Elements" for more information.*

Written out in a source document, the markup for the table in Figure 8-3 would look like the following sample. It is common to stack the **th** and **td** elements in order to make them easier to find in the source. This does not affect how they are rendered by the browser.

```
<table>
   <tr>
      <th>Menu item</th>
      <th>Calories</th>
      <th>Fat (g)</th>
   </tr>
   <tr>
      <td>Chicken noodle soup</td>
      <td>120</td>
      <td>2</td>
   </tr>
   <tr>
      <td>Caesar salad</td>
      <td>400</td>
      <td>26</td>
   </tr>
</table>
```

Remember, all the content must go in cells—that is, within **td** or **th** elements. You can put any content in a cell: text, a graphic, or even another table.

Start and end **table** tags are used to identify the beginning and end of the tabular material. The **table** element may directly contain only some number of **tr** (row) elements. The only thing that can go in the **tr** element is some number of **td** or **th** elements. In other words, there may be no text content within the **table** and **tr** elements that isn't contained within a **td** or **th**.

Finally, Figure 8-4 shows how the table would look in a simple web page, as displayed by default in a browser. I know it's not exciting. Excitement happens in the CSS chapters. What is worth noting is that tables always start on new lines by default in browsers.

Obras protegidas por Direitos de Autor

Advanced Table Elements

The sample table in this section has been stripped down to its bare essentials to make its structure clear while you learn how tables work. It is worth noting, however, that there are other table elements and attributes that offer more complex semantic descriptions and improve the accessibility of tabular content. A thoroughly marked-up version of the sample table might look like this:

```
<table>
<caption>Nutritional Information (Calorie and Fat
    Content)</caption>

<col span="1" class="itemname">
<colgroup id="data">
  <col span="1" class="calories">
  <col span="1" class="fat">
</colgroup>

<thead>
  <tr>
    <th scope="col">Menu item</th>
    <th scope="col">Calories</th>
    <th scope="column">Fat (g)</th>
  </tr>
</thead>

<tbody>
  <tr>
    <td>Chicken noodle soup</td>
    <td>120</td>
    <td>2</td>
  </tr>
  <tr>
    <td>Caesar salad</td>
    <td>400</td>
    <td>26</td>
  </tr>
</tbody>

</table>
```

Row group elements

You can describe rows or groups of rows as belonging to a header, footer, or the body of a table using the **`thead`**, **`tfoot`**, and **`tbody`** elements, respectively. Some user agents (another word for a browsing device) may repeat the header and footer rows on tables that span multiple pages. Authors may also use these elements to apply styles to various regions of a table.

Column group elements

Columns may be identified with the **`col`** element or put into groups using the **`colgroup`** element. This is useful for adding semantic context to information in columns and may be used to calculate the width of tables more quickly. Notice that there is no content in the column elements; it just describes the columns before the actual table data begins.

Accessibility features

Accessibility features such as captions for providing descriptions of table content and the **`scope`** and **`headers`** attributes for explicitly connecting headers with their respective content are discussed later in this chapter.

An in-depth exploration of the advanced table elements is beyond the scope of this book, but you may want to do more research at the W3C site (*www.w3.org/TR/html5*) if you anticipate working with data-heavy tables.

NOTE

According to the HTML5 spec, a table may contain "in this order: optionally a `caption` *element, followed by zero or more* `colgroup` *elements, followed optionally by a* `thead` *element, followed optionally by a* `tfoot` *element, followed by either zero or more* `tbody` *elements or one or more* `tr` *elements, followed optionally by a* `tfoot` *element (but there can only be one* `tfoot` *element child in total)." Got all that?*

Obras protegidas por Direitos de Autor

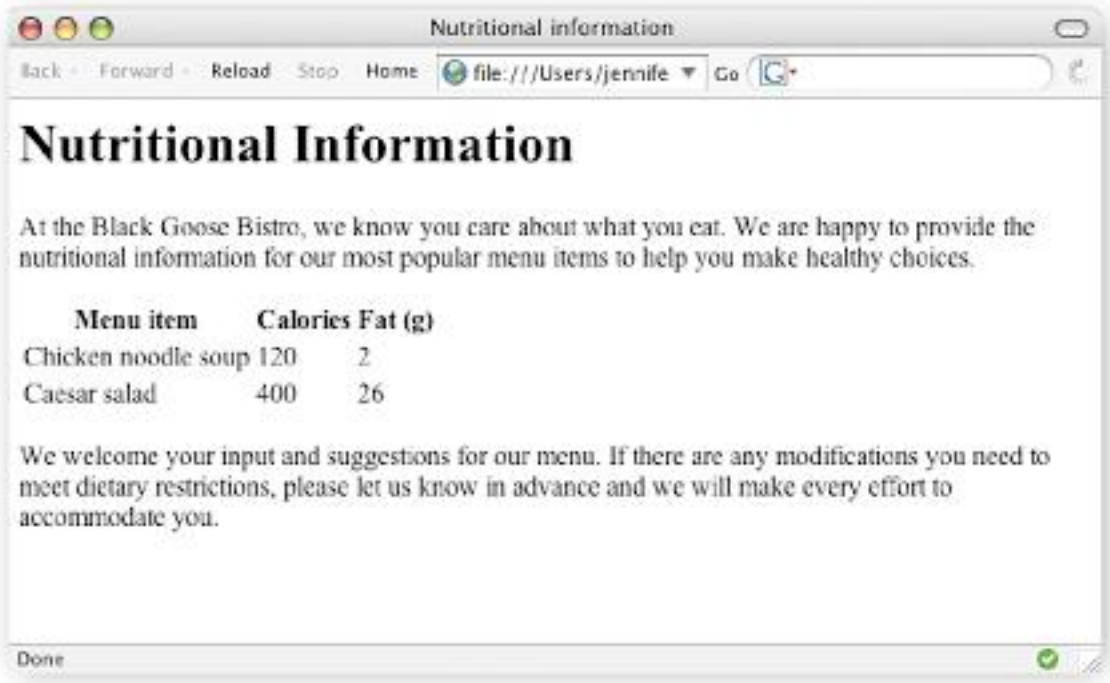


Figure 8-4. The default rendering of our sample table in a browser.

Here is the source for another table. Can you tell how many rows and columns it will have when it is displayed in a browser?

```
<table>
   <tr>
      <td>Sufjan Stevens</td>
      <td>Illinoise</td>
      <td>Asthmatic Kitty Records</td>
   </tr>
   <tr>
      <td>The Shins</td>
      <td>Oh Inverted World</td>
      <td>Sub-pop Records</td>
   </tr>
</table>
```

If you guessed that it's a table with two rows and three columns, you're correct! Two **tr** elements create two rows; three **td** elements in each row create three columns.

Table Headers

As you can see in Figure 8-4, the text marked up as headers (**th** elements) is displayed differently from the other cells in the table (**td** elements). The difference, however, is not purely cosmetic. Table headers are important because they provide information or context about the cells in the row or column they precede. The **th** element may be handled differently than **td**s by alternative browsing devices. For example, screen readers may read the header aloud before each data cell ("Menu item, Caesar salad, Calories, 400, Fat-g, 26").

Stylin' Tables

Once you build the structure of the table in the markup, it's no problem adding a layer of style to customize its appearance.

Style sheets can and should be used to control these aspects of a table's visual presentation. We'll get to all the formatting tools you'll need in the following chapters:

In Chapter 12, Formatting Text:

- Font settings for cell contents
- Text color in cells

In Chapter 14, Thinking Inside the Box:

- Table dimensions (width and height)
- Borders
- Cell padding (space around cell contents)
- Margins around the table

In Chapter 13, Colors and Backgrounds:

- Background colors
- Tiling background images

In Chapter 18, CSS Techniques:

- Special properties for controlling borders and spacing between cells

Obras protegidas por Direitos de Autor

In this way, they are a key tool for making table content accessible. Don't try to fake headers by formatting a row of `td` elements differently than the rest of the table. Conversely, don't avoid using `th` elements because of their default rendering (bold and centered). Mark up the headers semantically and change the presentation later with a style rule.

That covers the basics. Before we get fancier, try your hand at Exercise 8-1.

exercise 8-1 | Making a simple table

Try writing the markup for the table shown in Figure 8-5. You can open a text editor or just write it down on paper. The finished markup is provided in Appendix A.

(Note that I've added a 1-pixel border around cells with a style rule just to make the structure clear. You won't include this in your version.)

Be sure to close all table elements. Technically, you are not *required* to close `tr`, `th`, and `td` elements, but I want you to get in the habit of writing tidy source code for maximum predictability across all browsing devices. If you choose to write documents using XHTML syntax, closing table elements is required in order for the document to be valid.

Album	Year
Rubber Soul	1968
Revolver	1966
Sgt. Pepper's	1967
The White Album	1968
Abbey Road	1969

Figure 8-5. Write the markup for this table.

Spanning Cells

One fundamental feature of table structure is cell spanning, which is the stretching of a cell to cover several rows or columns. Spanning cells allows you to create complex table structures, but it has the side effect of making the markup a little more difficult to keep track of. You make a header or data cell span by adding the `colspan` or `rowspan` attributes, as we'll discuss next.

Obras protegidas por Direitos de Autor

Column spans

Column spans, created with the **colspan** attribute in the **td** or **th** element, stretch a cell to the right to span over the subsequent columns (Figure 8-6). Here a column span is used to make a header apply to two columns. (I've added a border around cells to reveal the table structure in the screenshot.)

WARNING

Be careful with `colspan` *values. If you specify a number that exceeds the number of columns in the table, most browsers will add columns to the existing table, which typically screws things up.*

```
<table>
   <tr>
      <th colspan="2">Fat</th>
   </tr>
   <tr>
      <td>Saturated Fat (g)</td>
      <td>Unsaturated Fat (g)</td>
   </tr>
</table>
```

Fat	
Saturated Fat (g)	Unsaturated Fat (g)

Figure 8-6. The `colspan` *attribute stretches a cell to the right to span the specified number of columns.*

Notice in the first row (**tr**) that there is only one **th** element, while the second row has two **td** elements. The **th** for the column that was spanned over is no longer in the source; the cell with the **colspan** stands in for it. Every row should have the same number of cells or equivalent **colspan** values. For example, there are two **td** elements and the **colspan** value is 2, so the implied number of columns in each row is equal.

exercise 8-2 | Column spans

Try writing the markup for the table shown in Figure 8-7. You can open a text editor or just write it down on paper. I added borders to reveal the cell structure in the figure, but your table won't have them. Check Appendix A for the final markup.

Some hints:

- For simplicity's sake, this table uses all **td** elements.
- The second row shows you that the table has a total of three columns.
- When a cell is spanned over, its **td** element does not appear in the table.

7:00pm	7:30pm	8:00pm
The Sunday Night Movie		
Perry Mason	Candid Camera	What's My Line
Bonanza	The Wackiest Ship in the Army	

Figure 8-7. Practice column spans by writing the markup for this table.

Obras protegidas por Direitos de Autor

Row spans

Row spans, created with the `rowspan` attribute, work just like column spans, but they cause the cell to span downward over several rows. In this example, the first cell in the table spans down three rows (Figure 8-8).

```
<table>
   <tr>
      <th rowspan="3">Serving Size</th>
      <td>Small (8oz.)</td>
   </tr>
   <tr>
      <td>Medium (16oz.)</td>
   </tr>
   <tr>
      <td>Large (24oz.)</td>
   </tr>
</table>
```

Again, notice that the `td` elements for the cells that were spanned over (the first cells in the remaining rows) do not appear in the source. The `rowspan="3"` implies cells for the subsequent two rows, so no `td` elements are needed.

Serving Size	Small (8oz.)
	Medium (16oz.)
	Large (24oz.)

Figure 8-8. The `rowspan` attribute stretches a cell downward to span the specified number of rows.

exercise 8-3 | Row spans

Try writing the markup for the table shown in Figure 8-9. Remember that cells that are spanned over do not appear in the table code. Rows always span downward, so the "oranges" cell is part of the first row even though its content is vertically centered.

If you're working in a text editor, don't worry if your table doesn't look exactly like the one shown here. The resulting markup is provided in Appendix A.

Some hints:

- Rows always span downward, so the "oranges" cell is part of the first row.
- Cells that are spanned over do not appear in the code.

apples	oranges	pears
bananas		pineapple
lychees		

Figure 8-9. Practice row spans by writing the markup for this table.

Obras protegidas por Direitos de Autor

Space in and Between Cells

By default, cells are sized just large enough to fit their contents, but often you'll want to add a little breathing room around tabular content (Figure 8-10). Because spacing is a matter of presentation, it is a job for style sheets.

Cell padding is the space inside the cell, between the content and the edge of the cell. To add cell padding, apply the CSS **`padding`** property to the **`td`** or **`th`** element.

Cell spacing, the area between cells, is a little more complicated. First, set the **`border-collapse`** property for the **`table`** to **`separate`**, then use the **`border-spacing`** property to specify the amount of space between borders. Unfortunately, this technique won't work in Internet Explorer 6, but hopefully IE6 usage will be inconsequential by the time you're reading this.

In the past, cell padding and spacing were handled by the **`cellpadding`** and **`cellspacing`** attributes in the **`table`** element, respectively, but they have been made obsolete in HTML5 due to their presentational nature.

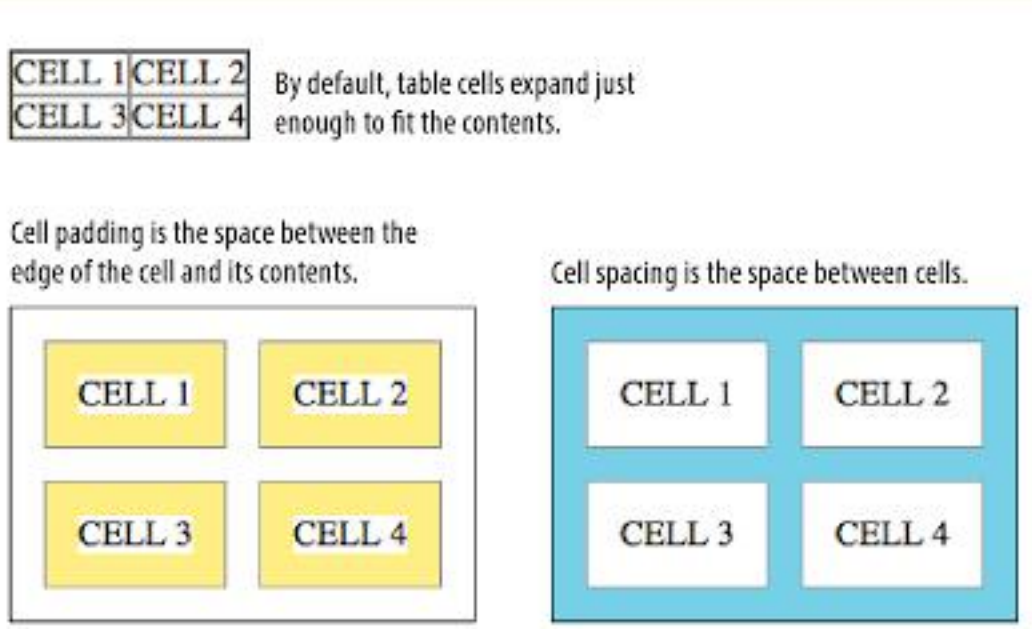


Figure 8-10. Cell padding and cell spacing.

Table Accessibility

As a web designer, it is important that you always keep in mind how your site's content is going to be used by non-sighted visitors. It is especially challenging to make sense of tabular material using a screen reader, but there are measures you can take to improve the experience and make your content more understandable.

Obras protegidas por Direitos de Autor

image not available

scope

The **scope** attribute associates a table header with the row, column, group of rows (such as **tbody**), or column group in which it appears using the values **row**, **column**, **rowgroup**, or **colgroup**, respectively. This example uses the **scope** attribute to declare that a header cell applies to the current row.

```
<tr>
   <th scope="row">Mars</th>
   <td>.95</td>
   <td>.62</td>
   <td>0</td>
</tr>
```

headers

For really complicated tables in which **scope** is not sufficient to associate a table data cell with its respective header (such as when the table contains multiple spanned cells), the **headers** attribute is used in the **td** element to explicitly tie it to a header's **id** value. In this example, the cell content ".38" is tied to the header "Diameter measured in earths":

```
<th id="diameter">Diameter measured in earths</th>

…many other cells…
<td headers="diameter">.38</td>
…many other cells…
```

This section is obviously only the tip of the iceberg of table accessibility. In-depth instruction on authoring accessible tables is beyond the scope of this beginner book. If you'd like to learn more, I recommend "Creating Accessible Tables" at WebAIM (*www.webaim.org/techniques/tables*) as an excellent starting point.

Wrapping Up Tables

This chapter gave you a good overview of the components of HTML tables. Exercise 8-4 puts most of what we covered together to give you a little more practice at authoring tables.

After just a few exercises, you're probably getting the sense that writing table markup manually, although not impossible, gets tedious and complicated quickly. Fortunately, web-authoring tools such as Dreamweaver provide interfaces that make the process much easier and time-efficient. Still, you'll be glad that you have a solid understanding of table structure and terminology, as well as the preferred methods for changing a table's appearance.

Obras protegidas por Direitos de Autor

exercise 8-4 | The table challenge

Now it's time to put together the table writing skills you've acquired in this chapter. Your challenge is to write out the source document for the table shown in Figure 8-12.

I'll walk you through it one step at a time.

1. First, open a new document in your text editor and set up its overall structure (**html**, **head**, **title**, and **body** elements). Save the document as *table.html* in the directory of your choice.
2. Next, in order to make the boundaries of the cells and table clearer when you check your work, I'm going to have you add some simple style sheet rules to the document. Don't worry about understanding exactly what's happening here (although it's fairly intuitive); just insert this **style** element in the **head** of the document exactly as you see it here.

```
<head>
  <title>Table Challenge</title>
  <style type="text/css">
    td, th { border: 1px solid #CCC; }
    table {border: 1px solid black; }
  </style>
</head>
```

3. Now it's time to start building the table. I usually start by setting up the table and adding as many empty row elements as I'll need for the final table as placeholders, as shown here (it should be clear that there are five rows in this table).

```
<body>
<table>
  <tr></tr>
  <tr></tr>
  <tr></tr>
  <tr></tr>
  <tr></tr>
</table>
</body>
```

4. Start with the top row, and fill in the **th** and **td** elements from left to right, including any row or column spans as necessary. I'll help with the first row.

   The first cell (the one in the top-left corner) spans down the height of two rows, so it gets a **rowspan** attribute. I'll use a **th** here to keep it consistent with the rest of the row. This cell has no content.

```
<table>
  <tr>
    <th rowspan="2"></th>
  </tr>
```

   The cell in the second column of the first row spans over the width of two columns, so it gets a **colspan** attribute:

```
<table>
  <tr>
    <th rowspan="2"></th>
    <th colspan="2">A common header for two
    subheads</th>
  </tr>
```

   The cell in the third column has been spanned over by the **colspan** we just added, so we don't need to include it in the markup. The cell in the fourth column also spans down two rows.

```
<table>
  <tr>
    <th rowspan="2"></th>
    <th colspan="2">A common header for two
    subheads</th>
    <th rowspan="2">Header 3</th>
  </tr>
```

5. Now it's your turn. Continue filling in the **th** and **td** elements for the remaining four rows of the table. Here's a hint: the first and last cells in the second row have been spanned over. Also, if it's bold in the example, make it a header.
6. To complete the content, add the title over the table using the **caption** element.
7. Finally, use the **scope** attribute to make sure that the Thing A, Thing B, and Thing C headers are associated with their respective rows.
8. Save your work and open the file in a browser. The table should look just like the one on this page. If not, go back and adjust your markup. If you're stumped, the final markup for this exercise is listed in Appendix A.

Your Content Here

	A common header for two subheads		Header 3
	Header 1	Header 2	
Thing A	data A1	data A2	data A3
Thing B	data B1	data B2	data B3
Thing C	data C1	data C2	data C3

Figure 8-12. The table challenge.

Obras protegidas por Direitos de Autor

image not available

FORMS

It didn't take long for the Web to shift from a network of pages to read to a place where you went to get things *done*—making purchases, booking plane tickets, signing petitions, searching a site, posting a tweet...the list goes on! All of these interactions are handled by forms.

In fact, in response to this shift from page to application, HTML5 introduced a bonanza of new form controls and attributes that make it easier for users to fill out forms and for developers to create them. Tasks that have traditionally relied on JavaScript may be handled by markup and native browser behavior alone. HTML5 introduces a number of new form-related elements, 13 new input types, and many new attributes (they are listed in Table 9-1 at the end of this chapter). Some of these features are waiting for browser implementation to catch up, so I will be sure to note which controls may not be universally supported.

This chapter introduces web forms, how they work, and the markup used to create them. I'll also briefly discuss the importance of web form design.

IN THIS CHAPTER

How forms work
The `form` element
POST versus GET
Variables and values
Form controls
Form accessibility features

How Forms Work

There are two parts to a working form. The first part is the form that you see on the page itself that is created using HTML markup. Forms are made up of buttons, input fields, and drop-down menus (collectively known as form controls) used to collect information from the user. Forms may also contain text and other elements.

The other component of a web form is an application or script on the server that processes the information collected by the form and returns an appropriate response. It's what makes the form *work*. In other words, posting an HTML document with form elements isn't enough. Web applications and scripts require programming know-how that is beyond the scope of this book, but the "Getting Your Forms to Work" sidebar, later in this chapter, provides some options for getting the scripts you need.

Obras protegidas por Direitos de Autor

A Word About Encoding

Form data is encoded using the same method used for URLs in which spaces and other characters that are not permitted are translated into their hexadecimal equivalents. For example, each space character in the collected form data is represented by the character string `%20`, and a slash (/) character is replaced with `%2F`. You don't need to worry about this; the browser handles it automatically.

From data entry to response

If you are going to be creating web forms, it is beneficial to understand what is happening behind the scenes. This example traces the steps of a transaction using a simple form that gathers names and email addresses for a mailing list; however, it is typical of the process for many forms.

1. Your visitor, let's call her Sally, opens the page with a web form in the browser window. The browser sees the form control elements in the markup and renders them with the appropriate form controls on the page, including two text entry fields and a submit button (shown in Figure 9-1).
2. Sally would like to sign up for this mailing list, so she enters her name and email address into the fields and submits the form by hitting the "Submit" button.
3. The browser collects the information she entered, encodes it (see the sidebar "A Word About Encoding"), and sends it to the web application on the server.

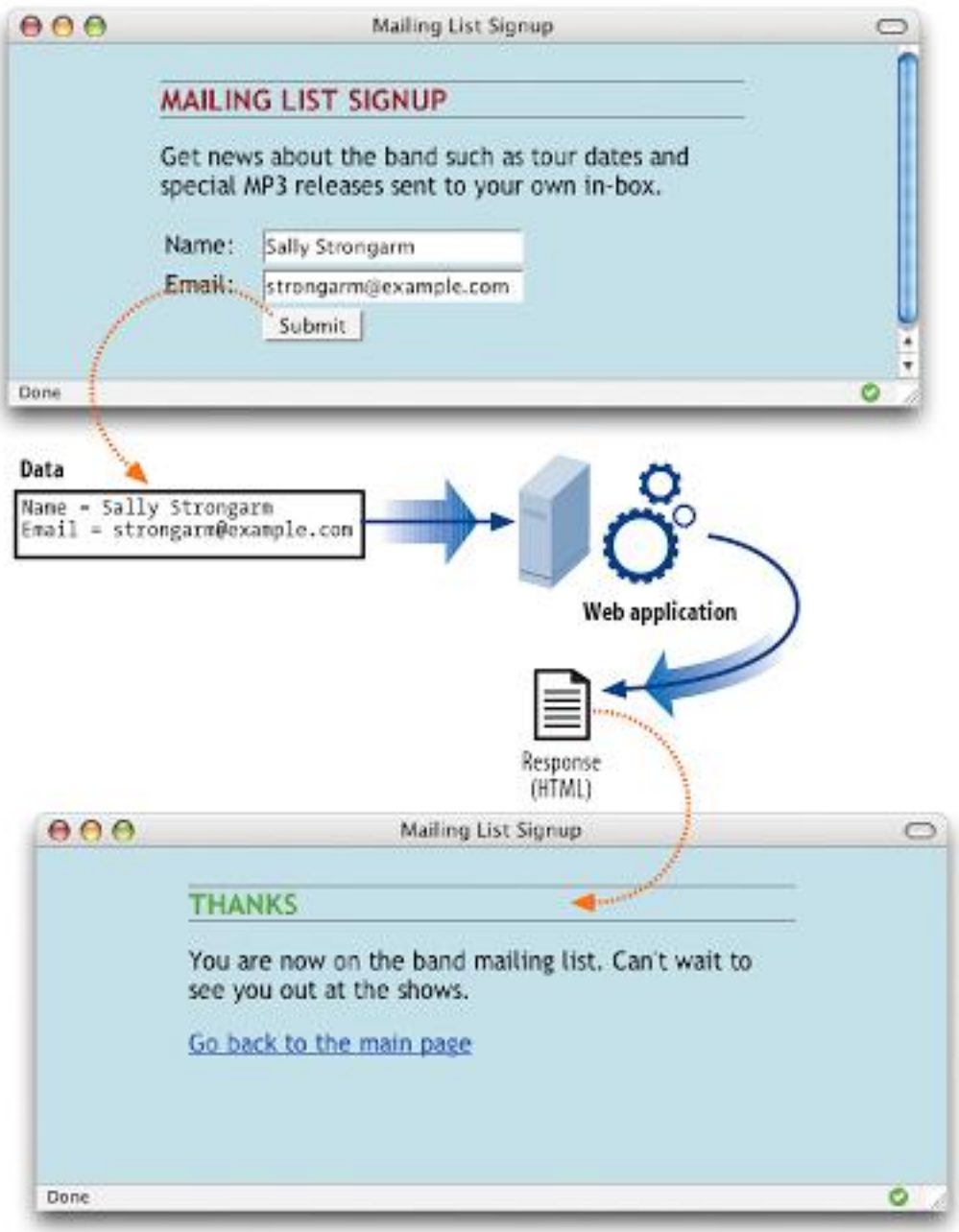


Figure 9-1. What happens behind the scenes when a web form is submitted.

4. The web application accepts the information and processes it (that is, does whatever it is programmed to do with it). In this example, the name and email address are added to a database.
5. The web application also returns a response. The kind of response sent back depends on the content and purpose of the form. Here, the response is a simple web page that contains a thank you for signing up for the mailing list. Other applications might respond by reloading the HTML form page with updated information, by moving the user on to another related form page, or by issuing an error message if the form is not filled out correctly, to name only a few examples.
6. The server sends the web application's response back to the browser where it is displayed. Sally can see that the form worked and that she has been added to the mailing list.

Obras protegidas por Direitos de Autor

The form Element

`<form>...</form>`

Interactive form

Forms are added to web pages using (no surprise here) the **form** element. The **form** element is a container for all the content of the form, including some number of form controls, such as text entry fields and buttons. It may also contain block elements (**h1**, **p**, and lists, for example). However, it may *not* contain another **form** element.

TIP

Be careful not to nest **form** elements or allow them to overlap. A **form** element must be closed before the next one begins.

This sample source document contains a form similar to the one shown in Figure 9-1:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mailing List Signup</title>
  <meta charset="utf-8">
</head>
<body>
  <h1>Mailing List Signup</h1>

  <form action="/mailinglist.php" method="post">
    <fieldset>
    <legend>Join our email list</legend>
    <p>Get news about the band such as tour dates and special MP3
releases sent to your own in-box.</p>
    <ol>
      <li><label for="firstlast">Name:</label>
          <input type="text" name="username" id="firstlast"></li>
      <li><label for="email">Email:</label>
          <input type="text" name="email" id="email"></li>
      </ol>
      <input type="submit" value="Submit">
      </fieldset>
  </form>

</body>
</html>
```

NOTE

It is current best practice to wrap form controls in semantic HTML elements such as lists or **divs**. *Ordered lists, as shown in this example, are a popular solution, but know that there are often default styles that need to be cleared out before styling them, particularly on mobile browsers.*

In addition to being a container for form control elements, the **form** element has some attributes that are necessary for interacting with the form-processing program on the server. Let's take a look at each.

The action attribute

The **action** attribute provides the location (URL) of the application or script (sometimes called the action page) that will be used to process the form. The **action** attribute in this example sends the data to a script called *mailinglist.php*.

```
<form action="/mailinglist.php" method="post">...</form>
```


Obras protegidas por Direitos de Autor

Getting Your Forms to Work

If you aren't a programmer, don't fret. You have a few options for getting your forms operational.

Use hosting plan goodies

Many site hosting plans include access to scripts for simple functions such as mailing lists. More advanced plans may even provide everything you need to add a full shopping cart system to your site as part of your monthly hosting fee. Documentation or a technical support person should be available to help you use them.

Hire a programmer

If you need a custom solution, you may need to hire a programmer who has server-side programming skills. Tell your programmer what you are looking to accomplish with your form and he or she will suggest a solution. Again, you need to make sure you have permission to install scripts on your server under your current hosting plan, and that the server supports the language you choose.

The *.php* suffix indicates that this form is processed by a script written in the PHP scripting language, but web forms may be processed using one of the following technologies:

- PHP (*.php*) is an open source scripting language most commonly used with the Apache web server.
- Microsoft's ASP.NET (Active Server Pages) (*.asp*) is a programming environment for the Microsoft Internet Information Server (IIS).
- Ruby on Rails. Ruby is the programming language that is used with the Rails platform. Many popular web applications are built with it.
- JavaServer Pages (*.jsp*) is a Java-based technology similar to ASP.
- Python is a popular scripting language for web and server applications.

There are other forms-processing options that may have their own suffixes or none at all (as is the case for the Ruby on Rails platform). Check with your programmer, server administrator, or script documentation for the proper name and location of the program to be provided by the **`action`** attribute.

Sometimes there is form-processing code such as PHP embedded right in the HTML file. In that case, leave the action empty and the form will post to the page itself.

The method attribute

The **`method`** attribute specifies how the information should be sent to the server. Let's use this data gathered from the sample form in Figure 9-1 as an example.

```
username = Sally Strongarm
email = strongarm@example.com
```

When the browser encodes that information for its trip to the server, it looks like this (see the earlier sidebar if you need a refresher on encoding):

```
username=Sally%20Strongarm&email=strongarm%40example.com
```

There are only two methods for sending this encoded data to the server: POST or GET, indicated using the **`method`** attribute in the **`form`** element. The method is optional and will default to GET if omitted. We'll look at the difference between the two methods in the following sections. Our example uses the POST method, as shown here:

```
<form action="/cgi-bin/mailinglist.pl" method="POST">...</form>
```

The POST method

When the form's method is set to POST, the browser sends a separate server request containing some special headers followed by the data. Only the server sees the content of this request, thus it is the best method for sending secure information such as credit card numbers or other personal information.

Obras protegidas por Direitos de Autor

The POST method is also preferable for sending a lot of data, such as a lengthy text entry, because there is no character limit as there is for GET.

The GET method

With the GET method, the encoded form data gets tacked right onto the URL sent to the server. A question mark character separates the URL from the following data, as shown here:

```
get http://www.bandname.com/cgi-bin/mailinglist.pl?name=Sally%20Strongar
m&email=strongarm%40example.com
```

The GET method is appropriate if you want users to be able to bookmark the results of a form submission (such as a list of search results). Because the content of the form is in plain sight, GET is not appropriate for forms with private personal or financial information. In addition, GET may not be used when the form is used to upload a file.

In this chapter, we'll stick with the more prevalent POST method. Now that we've gotten through the technical aspects of the `form` element, we can take on the real meat of forms: form controls.

NOTE

POST and GET are not case-sensitive and are commonly listed in all uppercase by convention. In XHTML documents, however, the value of the `method` *attribute (post or get) must be provided in all lowercase letters.*

Variables and Content

Web forms use a variety of controls that allow users to enter information or choose options. Control types include various text entry fields, buttons, menus, and a few controls with special functions. They are added to the document using a collection of form control elements that we'll be examining one by one in the upcoming "Great Form Control Roundup" section.

As a web designer, you need to be familiar with control options to make your forms easy and intuitive to use. It is also useful to have an idea of what form controls are doing behind the scenes.

The name attribute

The job of a form control is to collect one bit of information from a user. In the form example a few pages back, text entry fields collect the visitor's name and email address. To use the technical term, "username" and "email" are two variables collected by the form. The data entered by the user ("Sally Strongarm" and "strongarm@example.com") is the value or content of the variable.

The **`name`** attribute provides the variable name for the control. In this example, the text gathered by a **`textarea`** element is defined as the "comment" variable:

```
<textarea name="comment" rows="4" cols="45" placeholder="Leave us a
comment."></textarea>
```


Obras protegidas por Direitos de Autor

When a user enters a comment in the field ("This is the best band ever!"), it would be passed to the server as a name/value (variable/content) pair like this:

```
comment=This%20is%20the%20best%20band%20ever%21
```

All form control elements must include a `name` attribute so the form-processing application can sort the information. You may include a `name` attribute for `submit` and `reset` button elements, but they are not required, because they have special functions (submitting or resetting the form) not related to data collection.

Naming your variables

You can't just name controls willy-nilly. The web application that processes the data is programmed to look for specific variable names. If you are designing a form to work with a preexisting application or script, you need to find out the specific variable names to use in the form so they are speaking the same language. You can get the variable names from the developer you are working with, your system administrator, or from the instructions provided with a ready-to-use script on your server.

If the script or application will be created later, be sure to name your variables simply and descriptively and to document them well. In addition, each variable must be named uniquely—that is, the same name may not be used for two variables. You should also avoid putting character spaces in variable names; use an underscore or hyphen instead.

We've covered the basics of the `form` element and how variables are named. Now we can get to the real meat of form markup: the controls.

The Great Form Control Roundup

This is the fun part—playing with the markup that adds form controls to the page. This section introduces the elements used to create:

- Text entry controls
- Specialized text entry controls
- Submit and reset buttons
- Radio and checkbox buttons
- Pull-down and scrolling menus
- File selection and upload control
- Hidden controls
- Dates and times (HTML5)
- Numerical controls (HTML5)
- Color picker control (HTML5)

Obras protegidas por Direitos de Autor

We'll pause along the way to allow you to try them out by constructing the questionnaire form shown in Figure 9-2.

As you will see, the majority of controls are added to a form using the `input` element. The functionality and appearance of the `input` element changes based on the value of the `type` attribute in the tag. In HTML5, there are *twenty-three* different types of input controls. We'll take a look at them all.

NOTE

The attributes associated with each input type are listed in Table 9-1 at the end of this chapter.

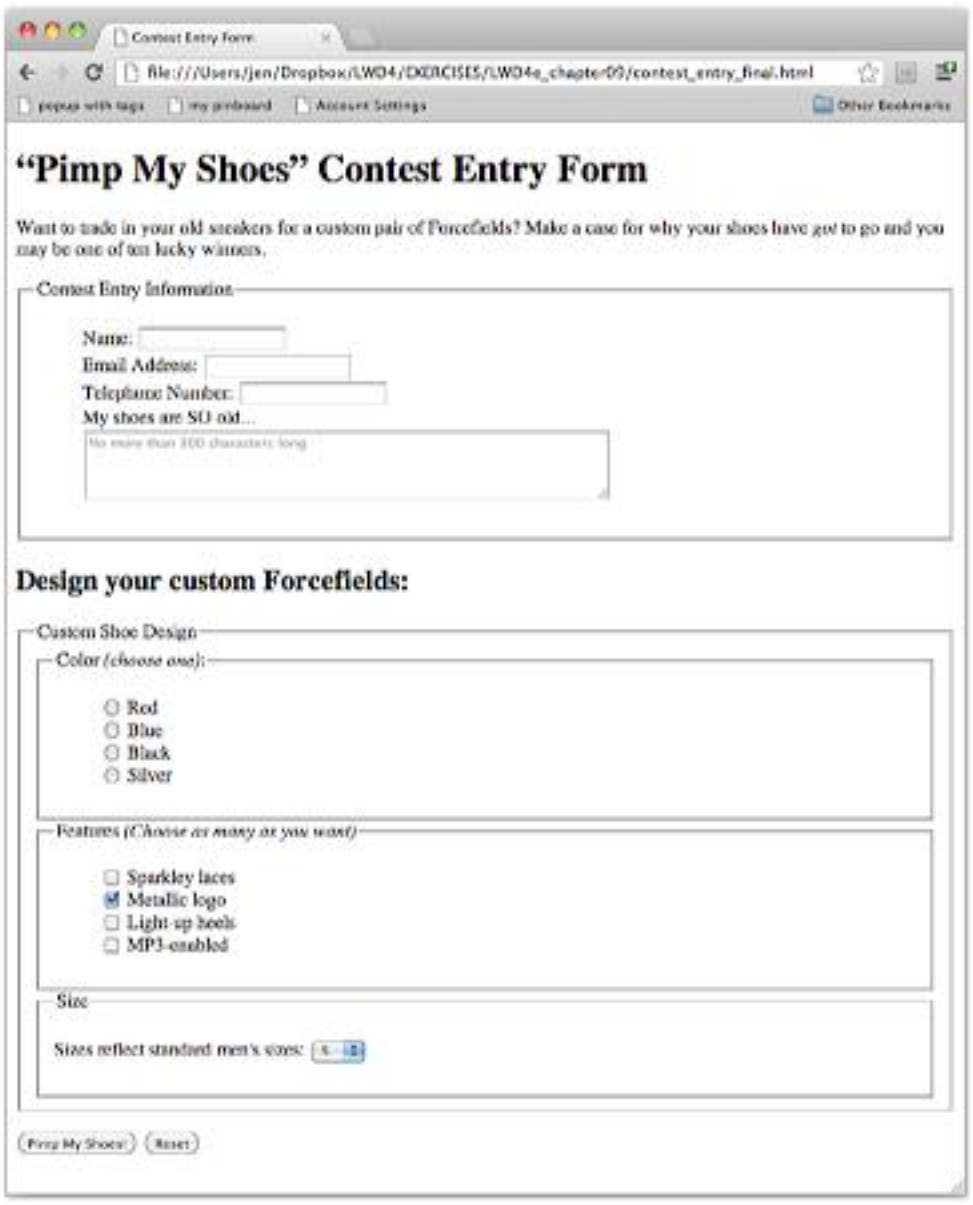


Figure 9-2. The contest entry form we'll be building in the exercises in this chapter.

Text entry controls

One of the most common tasks in a web form is to enter text information. Which element you use depends on whether users are asked to enter a single line of text (`input`) or multiple lines (`textarea`).

NOTE

The markup examples throughout this section include the `label` element, which is used to improve accessibility. We will discuss `label` in more detail in the "Form Accessibility Features" section later in this chapter, but in the meantime, I want you to get used to seeing proper form markup.

Obras protegidas por Direitos de Autor

Single-line text field

`<input type="text">`

Single-line text entry control

One of the most straightforward form input types is the text entry field used for entering a single word or line of text. In fact, it is the default input type, which means it is what you'll get if you forget to include the **type** attribute or include an unrecognized value. Add a text input field to a form with the **input** element with its **type** attribute set to **text**, as shown here and in Figure 9-3.

```
<li><label>City <input type="text" name="city" id="form-city"
value="Your Hometown" maxlength="50"></label></li>
```

There are a few attributes in there I'd like to point out.

`name`

The **name** attribute is required for indicating the variable name.

`value`

The **value** attribute specifies default text that appears in the field when the form is loaded. When you reset a form, it returns to this value.

`maxlength`

By default, users can type an unlimited number of characters in a text field regardless of its size (the display scrolls to the right if the text exceeds the character width of the box). You can set a maximum character limit using the **maxlength** attribute if the forms-processing program you are using requires it.

Multiline text entry field

`<textarea>...</textarea>`

Multiline text entry control

At times, you'll want your users to be able to enter more than just one line of text. For these instances, use the **textarea** element that is replaced by a multiline, scrollable text entry box when displayed by the browser (Figure 9-3).

NOTE

The specific rendering style of form controls varies by operating system and browser version.

Figure 9-3. Examples of the text entry control options for web forms.

Obras protegidas por Direitos de Autor

Unlike the empty **input** element, you can put content between the opening and closing tags in the **textarea** element. The content of the **textarea** element will show up in the text box when the form is displayed in the browser. It will also get sent to the server when the form is submitted, so carefully consider what goes there. It is not uncommon for developers to put nothing between the opening and closing tags, and provide a hint of what should go there with a **title** or **placeholder** attribute instead. The new HTML5 **placeholder** attribute can be used with **textarea** and other text-based **input** types and is used to provide a short hint of how to fill in the field. It is not supported on Android, older versions of Firefox (versions earlier than 3.6), or IE as of this writing.

```
<p><label>Official contest entry <br>
<em>Tell us why you love the band. Five winners will get backstage
passes!</em><br>
<textarea name="contest_entry" rows="5" cols="50">The band is totally
awesome!</textarea></label></p>

<p>Official contest entry:<br>
<em>Tell us why you love the band. Five winners will get backstage
passes!</em><br>
<textarea name="contest_entry" placeholder="50 words or less">
</textarea>
</p>
```

The **rows** and **cols** attributes are a way of specifying the size of the **textarea** using markup, but it is more commonly sized with CSS. **rows** specifies the number of lines the text area should display, and **cols** specifies the width in number of characters. Scrollbars will be provided if the user types more text than fits in the allotted space.

There are also a few attributes not shown in the example. The **wrap** attribute specifies whether the text should keep its line breaks when submitted. A value of **soft** (the default) does not preserve line breaks, and **hard** does. The **maxlength** attribute (new in HTML5) sets a limit on the number of characters that can by typed into the field.

Specialized text entry fields

In addition to the generic single-line text entry, there are a number of input types for entering specific types of information such as passwords, search terms, email addresses, telephone numbers, and URLs.

Password entry field

```
<input type="password">
```

Password text control

A password field works just like a text entry field, except the characters are obscured from view using asterisk (*) or bullet (•) characters, or another character determined by the browser.

disabled and readonly

The **disabled** and **readonly** attributes can be added to any form control element to prevent users from selecting them. When a form element is disabled, it cannot be selected. Visual browsers may render the control as grayed-out by default (which you can change with CSS, of course). The disabled state can only be changed with a script. This is a useful attribute for restricting access to some form fields based on data entered earlier in the form.

The **readonly** attribute prevents the user from changing the value of the form control (although it can be selected). This enables developers to use scripts to set values for controls contingent on other data entered earlier in the form. Inputs that are **readonly** should have strong visual cues that they are somehow different than other inputs, or they could be confusing to users who are trying to change their values.

WARNING

iOS ignores **disabled** *on* **option** *elements as of this writing (iOS 5 and earlier).*

Obras protegidas por Direitos de Autor

It's important to note that although the characters entered in the password field are not visible to casual onlookers, the form does not encrypt the information, so it should not be considered a real security measure.

Here is an example of the markup for a password field. Figure 9-4 shows how it might look after the user enters a password in the field.

```
<li><label for="form-pswd">Log in:</label><br>
 <input type="password" name="pswd" maxlength="8" id="form-pswd"></li>
```

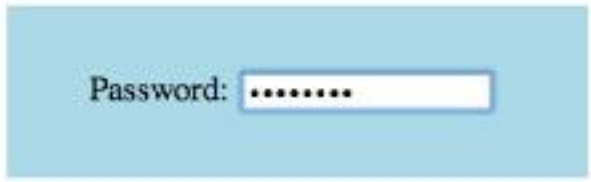


Figure 9-4. Passwords are converted to bullets in the browser display.

HTML5 text inputs

`<input type="search">`
Search field
NEW IN HTML5

`<input type="email">`
Email address
NEW IN HTML5

`<input type="tel">`
Telephone number
NEW IN HTML5

`<input type="url">`
Location (URL)
NEW IN HTML5

Until HTML5, the only way to collect email addresses, telephone numbers, URLs, or search terms was to insert a generic text input field. In HTML5, the **email**, **tel**, **url**, and **search** input types give the browser a heads-up as to what type of information to expect in the field. These new input types use the same attributes as the generic text input type described earlier (**name**, **maxlength**, **size**, and **value**), as well as a number of new HTML5 attributes.

All of these input types are typically displayed as single-line text inputs. But browsers that support them can do some interesting things with the extra semantic information. For example, Safari on iOS uses the input type to provide a keyboard well suited to the entry task, such as the keyboard featuring a "Search" button for the **search** input type or a ".com" button when the input type is set to **url** (Figure 9-5). Browsers usually add a one-click "clear field" icon (usually a little X) in search fields. A supporting browser could check the user's input to see that it is valid, such as making sure text entered in an **email** input follows standard email address structure (in the past, you needed JavaScript for validation). For example, the Opera (Figure 9-6) and Chrome browsers display a warning if the input does not match the expected format.

Not all browsers support the new HTML5 input types or support them in the same way, but the good news is that if the type isn't recognized, the default generic text input is displayed instead, which works perfectly fine. There is no reason not to start using them right away as a progressive enhancement, even if you can't reap the benefits of easy user input and browser (client-side) validation.

Obras protegidas por Direitos de Autor

Figure 9-5. Safari on iOS provides custom keyboards based on the input type.

Figure 9-6. Opera displays a warning when input does not match the expected `email` *format as part of its client-side validation support.*

The datalist Element

The **`datalist`** element (new in HTML5) allows the author to provide a drop-down menu of suggested values for any type of text input. It gives the user some shortcuts to select from, but if none are selected, the user can still type in her own text. Within the **`datalist`** element, suggested values are marked up as option elements. Use the **`list`** attribute in the **`input`** element to associate it with the **`id`** of its respective **`datalist`**.

In the following example (Figure 9-7), a **`datalist`** suggests several education level options for a text input.

```
<p>Education completed: <input type="text"
    list="edulevel" name="education">
<datalist id="edulevel">
  <option value="High School">
  <option value="Bachelors Degree">
  <option value="Masters Degree">
  <option value="PhD">
</datalist>
```

As of this writing, only the Opera browser has implemented the **`datalist`** element. Other browsers will ignore it and present a simple text input, which is a perfectly acceptable fallback. You could also use JavaScript to create **`datalist`** functionality (i.e., a polyfill).

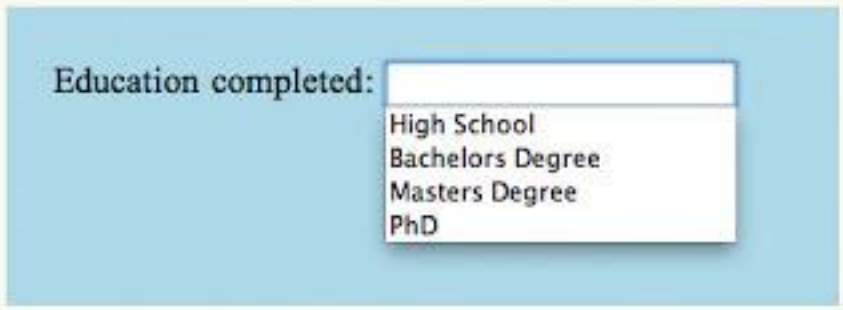


Figure 9-7. A `datalist` *creates a pop-up menu of suggested values for a text entry field.*

Obras protegidas por Direitos de Autor

Submit and reset buttons

`<input type="submit">`

Submits the form data to the server

`<input type="reset">`

Resets the form controls to their default settings

There are several different kinds of buttons that can be added to web forms. The most fundamental is the submit button. When clicked or tapped, the submit button immediately sends the collected form data to the server for processing. A reset button returns the form controls to the state they were in when the form initially loaded. In other words, resetting the form doesn't simply clear all the fields.

Both submit and reset buttons are added using the **input** element. As mentioned earlier, because these buttons have specific functions that do not include the entry of data, they are the only form control elements that do not require the **name** attribute, although it is OK to add one if you need it.

Submit and reset buttons are straightforward to use. Just place them in the appropriate place in the form, which in most cases is at the very end. By default, the submit button displays with the label "Submit" or "Submit Query" and the reset button is labeled "Reset." Change the text on the button using the **value** attribute, as shown in the reset button in this example (Figure 9-8).

```
<p><input type="submit"> <input type="reset" value="Start over"></p>
```

Figure 9-8. Submit and reset buttons.

The reset button is not used in forms as commonly as it used to be. That is because in contemporary form development, we use JavaScript to check the validity of form inputs along the way, so the users get feedback as they go along. With thoughtful design and assistance, fewer users should get to the end of the form and need to reset the whole thing. Still, it is a good function to be aware of.

At this point, you know enough about form markup to start building the questionnaire shown in Figure 9-2. Exercise 9-1 walks you through the first steps.

A Few More Buttons

There are a handful of custom button elements that are a little off the beaten path for beginners, but in the interest of thoroughness, here they are tucked off in a sidebar.

Image buttons

`<input type="image">`

This type of **input** control allows you to replace the submit button with an image of your choice. The image will appear flat, not like a 3-D button. Unfortunately, this type of button has accessibility issues, so be sure to include a carefully chosen **alt** value.

Custom input button

`<input type="button">`

Setting the type of the **input** element to "button" creates a button that can be customized with JavaScript. It has no predefined function on its own, unlike submit and reset buttons.

The button element

`<button>...</button>`

The **button** element is a flexible element for creating custom buttons similar to those created with the **input** element. The content of the button element (text and/or images) is what gets displayed on the button.

For more information on what you can do with the **button** element, read "Push My Button" by Aaron Gustafson at *digital-web.com/articles/push_my_button*.

Obras protegidas por Direitos de Autor

exercise 9-1 | Starting the contest form

Here's the scenario. You are the web designer in charge of creating the entry form for the Forcefield Sneakers "Pimp My Shoes!" Contest. The copy editor has handed you a sketch (Figure 9-9) of the form's content, complete with notes of how some controls should work. There are sticky notes from the programmer with information about the script and variable names you need to use.

Your challenge is to turn the sketch into a functional online form. I've given you a head start by creating a bare-bones document containing the text content and some minimal markup and styles. This document, ***contest_entry.html***, is available online at *www.learningwebdesign.com/4e/materials*. The source for the entire finished form is provided in Appendix A if you want to check your work.

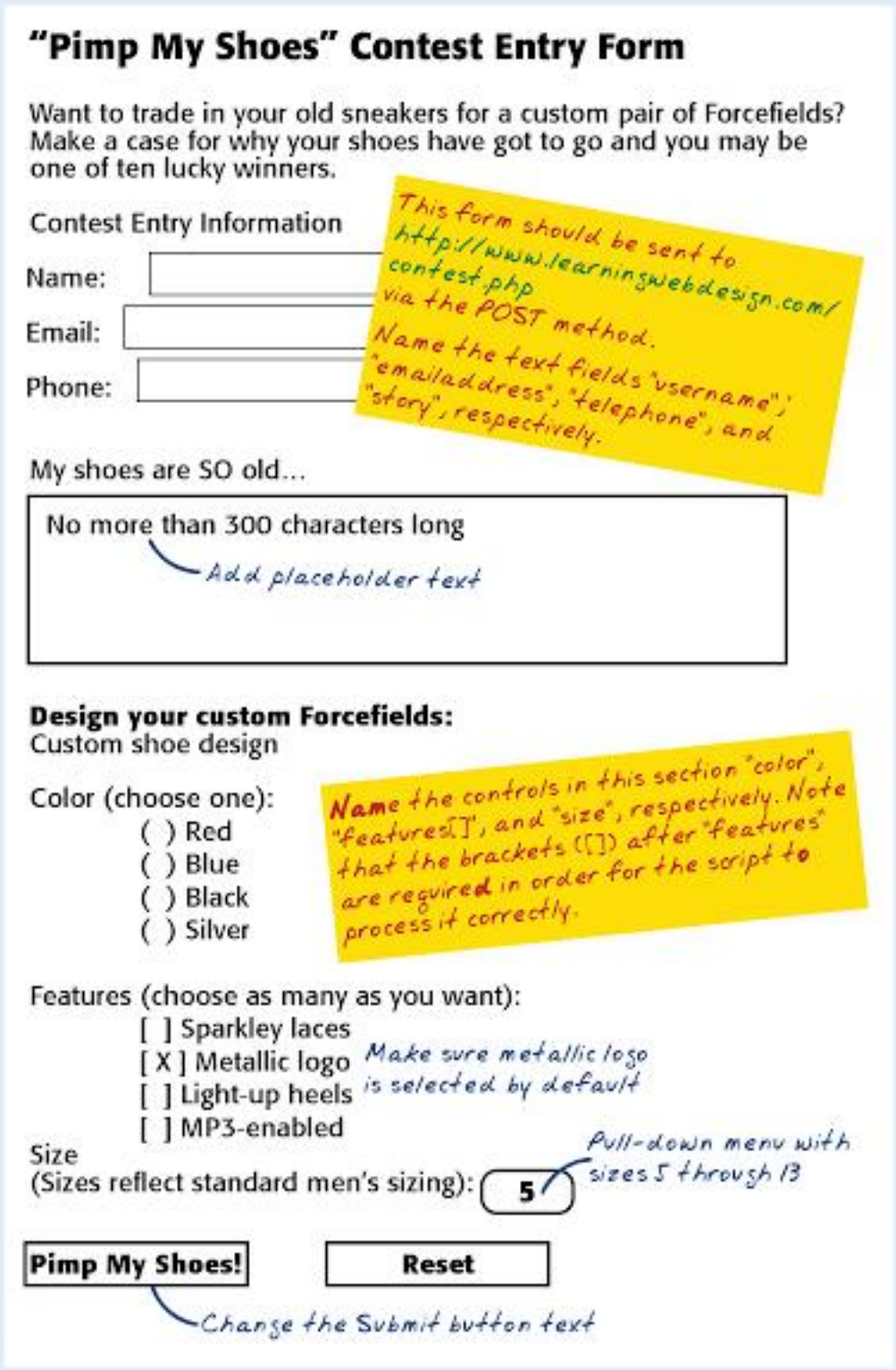


Figure 9-9. A sketch of the contest entry form.

Obras protegidas por Direitos de Autor

1. Open *contest_entry.html* in a text editor.
2. The first thing we'll do is put everything after the intro paragraph into a **form** element. The programmer has left a note specifying the **action** and the **method** to use for this form. The resulting **form** element should look like this:

```
<form action="http://www.learningwebdesign.com/contest.php"
method="post">
...
</form>
```

3. In this exercise, we'll work on the "Contest Entry Information" section of the form. Start with the first three short text entry form controls that are marked up appropriately as an unordered list. Here's the first one; you insert the other two.

```
<li>Name: <input type="text" name="username"></li>
```

   Hints: choose the most appropriate input type for each entry field. Be sure to name the input elements as specified in the programmer's note.

4. Now add a multiline text area for the shoe description on a new line. Because we aren't writing a style sheet for this form, use markup to make it four rows long and 60 characters wide (in the real world, CSS is preferable because it gives you more fine-tuned control).

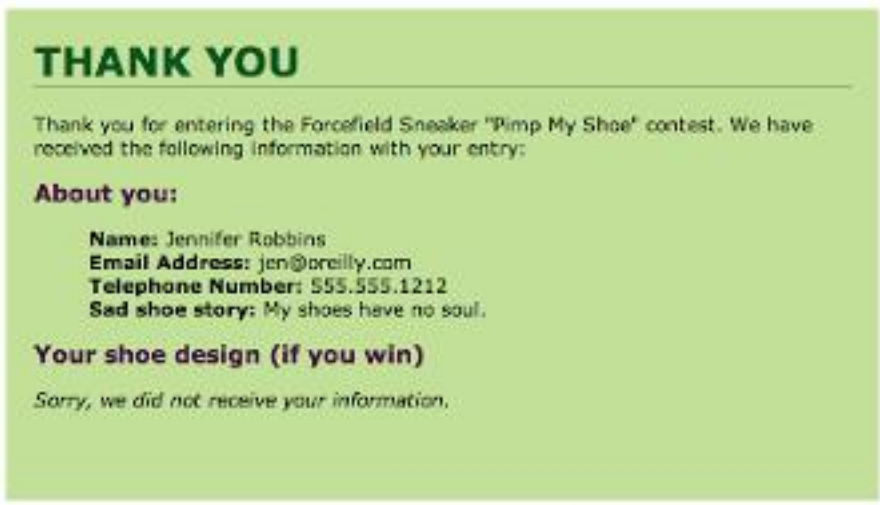
```
<li>My shoes are SO old...<br>
<textarea name="story" rows="4" cols="60"
maxlength="300" placeholder="No more than 300
characters long"></textarea></li>
```

5. We'll skip the rest of the form for now until we get a few more controls under our belt, but we can add the submit and reset buttons at the end, just before the **</form>** tag. Note that we need to change the text on the submit button.

```
<p><input type="submit" value="Pimp my shoes!">
<input type="reset"></p>
</form>
```

6. Now, save the document and open it in a browser. The parts that are finished should generally match Figure 9-3. If they don't, then you have some more work to do.

Once the document looks right, take it for a spin by entering some information and submitting the form. You should get a response like the one shown in Figure 9-10 (yes, ***contact.php*** actually works, but sorry, the contest is make-believe).



THANK YOU

Thank you for entering the Forcefield Sneaker "Pimp My Shoe" contest. We have received the following information with your entry:

About you:

Name: Jennifer Robbins
Email Address: jen@oreilly.com
Telephone Number: 555.555.1212
Sad shoe story: My shoes have no soul.

Your shoe design (if you win)

Sorry, we did not receive your information.

Figure 9-10. You should see a response page like this if your form is working.

Obras protegidas por Direitos de Autor

Radio and checkbox buttons

Both checkbox and radio buttons make it simple for your visitors to choose from a number of provided options. They are similar in that they function like little on/off switches that can be toggled by the user and are added using the **input** element. They serve distinct functions, however.

A form control made up of a collection of radio buttons is appropriate when only one option from the group is permitted—in other words, when the selections are mutually exclusive (such as Yes or No, or Male or Female). When one radio button is "on," all of the others must be "off," sort of the way buttons used to work on old radios: press one button in and the rest pop out.

When checkboxes are grouped together, however, it is possible to select as many or as few from the group as desired. This makes them the right choice for lists in which more than one selection is OK.

NOTE

I have omitted the `fieldset` *and* `label` *elements from the code examples for radio buttons, checkboxes, and menus in order to keep the markup structure as simple and clear as possible. In the upcoming "Form Accessibility Features" section, you will learn why it is important to include them in your markup for all form elements.*

Radio buttons

Radio buttons are added to a form using the **input** element with the **type** attribute set to **radio**. Here is the syntax for a minimal radio button:

```
<input type="radio" name="variable" value="value">
```

```
<input type="radio">
```

Radio button

The **name** attribute is required and plays an important role in binding multiple radio inputs into set. When you give a number of radio button inputs the same **name** value (**age** in the following example), they create a group of mutually exclusive options.

In this example, radio buttons are used as an interface for users to enter their age group (a person can't belong to more than one age group, so radio buttons are the right choice). Figure 9-11 shows how radio buttons are rendered in the browser.

```
<p>How old are you?</p>
<ol>
  <li><input type="radio" name="age" value="under24" checked> under
24</li>
  <li><input type="radio" name="age" value="25-34"> 25 to 34</li>
  <li><input type="radio" name="age" value="35-44"> 35 to 44</li>
  <li><input type="radio" name="age" value="over45"> 45+</li>
</ol>
```

Notice that all of the **input** elements have the same variable name ("age"), but their values are different. Because these are radio buttons, only one button can be checked at a time, and therefore, only one value will be sent to the server for processing when the form is submitted.

Obras protegidas por Direitos de Autor

NOTE

In XHTML syntax, the value of the **checked** *attribute must be explicitly set to* **checked**, *as shown in the example.*

```
<input type="radio" name="foo"
checked="checked" />
```

But in HTML syntax, you don't need to write out the value for the **checked** *attribute. It can be minimized, as shown here:*

```
<input type="radio" name="foo"
checked >
```

exercise 9-2 | Adding radio buttons and checkboxes

The next two questions in the sneaker contest entry form use radio buttons and checkboxes for selecting options. Open the *contest_entry.html* document and follow these steps.

1. In the Custom Shoe Design section, there are lists of color and feature options. The Color options should be radio buttons because shoes can be only one color. Insert a radio button before each option. Follow this example for the remaining color options.

   ```
   <li><input type="radio"
   name="color" value="red">
   Red</li>
   ```

2. Mark up the Features options as you did the Color options, but this time, the **type** should be **checkbox**. Be sure the variable name for each is **features[]**, and that the metallic logo option is preselected, as noted on the sketch.
3. Save the document and check your work by opening it in a browser to make sure it looks right, then submit the form to make sure it's functioning properly.

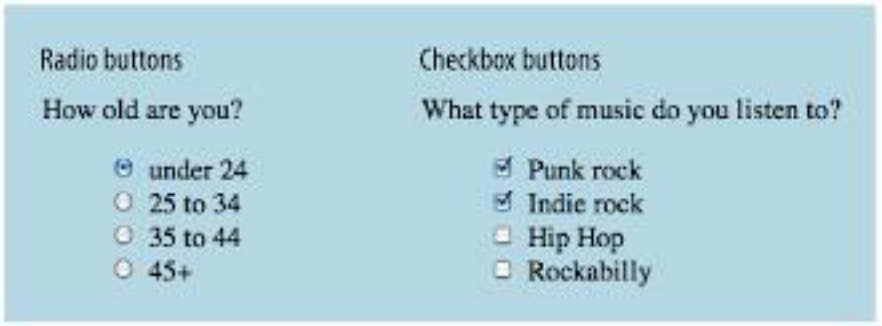


Figure 9-11. Radio buttons (left) are appropriate when only one selection is permitted. Checkboxes (right) are best when users may choose any number of choices, from none to all of them.

You can decide which button is checked when the form loads by adding the **checked** attribute to the **input** element. In this example, the button next to "under 24" will be checked by default (see the note).

Checkbox buttons

```
<input type="checkbox">
```

Checkbox button

Checkboxes are added using the **input** element with its type set to **checkbox**. As with radio buttons, you create groups of checkboxes by assigning them the same **name** value. The difference, as we've already noted, is that more than one checkbox may be checked at a time. The value of every checked button will be sent to the server when the form is submitted. Here is an example of a group of checkbox buttons used to indicate musical interests. Figure 9-11 shows how they look in the browser:

```
<p>What type of music do you listen to?</p>
<ul>
  <li><input type="checkbox" name="genre" value="punk" checked> Punk
rock</li>
  <li><input type="checkbox" name="genre" value="indie" checked> Indie
rock</li>
  <li><input type="checkbox" name="genre" value="hiphop"> Hip Hop</li>
  <li><input type="checkbox" name="genre" value="rockabilly">
Rockabilly</li>
</ul>
```

Checkboxes don't necessarily need to be used in groups, of course. In this example, a single checkbox is used to allow visitors to opt in for special promotions. The value of the control will be passed along to the server only if the user checks the box.

```
<p><input type="checkbox" name="OptIn" value="yes"> Yes, send me news
and special promotions by email.</p>
```

In Exercise 9-2, you'll get a chance to add both radio and checkbox buttons to the contest entry form.

Obras protegidas por Direitos de Autor

image not available

the same options as the previous one, except it has been set to display as a scrolling list that is six lines tall (Figure 9-13).

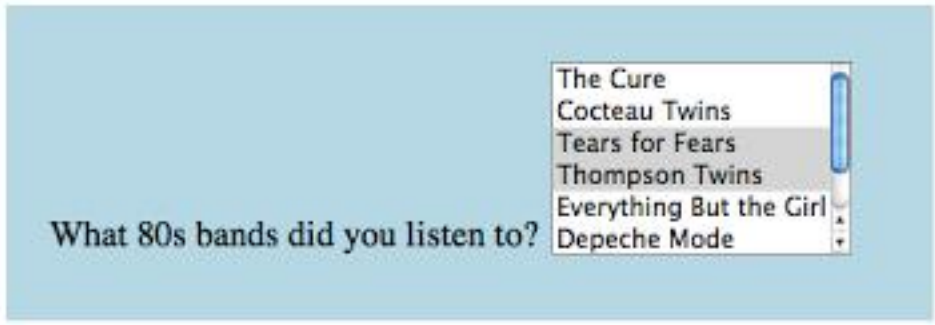


Figure 9-13. A scrolling menu with multiple options selected.

```
<p>What 80s bands did you listen to?
<select name="EightiesBands" size="6" multiple>
   <option>The Cure</option>
   <option>Cocteau Twins</option>
   <option selected>Tears for Fears</option>
   <option selected>Thompson Twins</option>
   <option value="EBTG">Everything But the Girl</option>
   <option>Depeche Mode</option>
   <option>The Smiths</option>
   <option>New Order</option>
</select>
</p>
```

You may notice a few new attributes tucked in there. The **multiple** attribute allows users to make more than one selection from the scrolling list. Note that pull-down menus do not allow multiple selections; when the browser detects the **multiple** attribute, it displays a small scrolling menu automatically by default.

Use the **selected** attribute in an **option** element to make it the default value for the menu control. Selected options are highlighted when the form loads. The **selected** attribute can be used with pull-down menus as well.

NOTE

The `label` *attribute in the* `option` *element is not the same as the* `label` *element used to improve accessibility (discussed later in this chapter).*

Grouping menu options

You can use the **optgroup** element to create conceptual groups of options. The required **label** attribute in the **optgroup** element provides the heading for the group. Figure 9-14 shows how option groups are rendered in modern browsers.

```
<select name="icecream" size="7" multiple>
  <optgroup label="traditional">
      <option>vanilla</option>
      <option>chocolate</option>
  </optgroup>
  <optgroup label="fancy">
    <option>Super praline</option>
    <option>Nut surprise</option>
    <option>Candy corn</option>
  </optgroup>
</select>
```


Obras protegidas por Direitos de Autor

Figure 9-14. Option groups as rendered in a modern browser.

exercise 9-3 | Adding a menu

The only other control that needs to be added to the contest entry is a pull-down menu for selecting a shoe size.

1. Insert a **select** menu element with the shoe sizes (5 to 13).

```
<p>Size (sizes reflect men's sizing):
  <select name="size" size="1">
    <option>5</option>
    ...insert more options here...
  </select>
</p>
```

2. Save the document and check it in a browser. You can submit the form, too, to be sure that it's working. You should get the Thank You response page listing all of the information you entered in the form.

Congratulations! You've built your first working web form. In Exercise 9-4, we'll add markup that makes it more accessible to assistive devices. But first, we have a few more control types to cover.

File selection control

```
<input type="file">
```

File selection field

Web forms can collect more than just data. They can also be used to transmit external documents from a user's hard drive. For example, a printing company could use a web form to upload artwork for a business card order. A magazine could use a form on its site to collect digital photos for a photo contest.

The file selection control makes it possible for users to select a document from the hard drive to be submitted with the form data. It is added to the form using our old friend, the **input** element, with its **type** set to **file**.

The markup sample here and Figure 9-15 show a file selection control used for photo submissions.

```
<form action="/client.php" method="POST" enctype="multipart/form-data">
  <label>Send a photo to be used as your online icon
  <em>(optional)</em><br>
  <input type="file" name="photo" size="28"><label>
</form>
```


Obras protegidas por Direitos de Autor

The file upload widget varies by browser and operating system. It may be a text field with a button to browse the hard drive, as Firefox does (Figure 9-15, top) or it might be just a button, which is how Chrome displays it (bottom).

It is important to note that when a form contains a file selection input element, you must specify the encoding type (`enctype`) of the form as `multipart/form-data` and use the POST method. The `size` attribute in this example sets the character width of the text field (although it could also be controlled with a CSS rule) if the browser displays one.

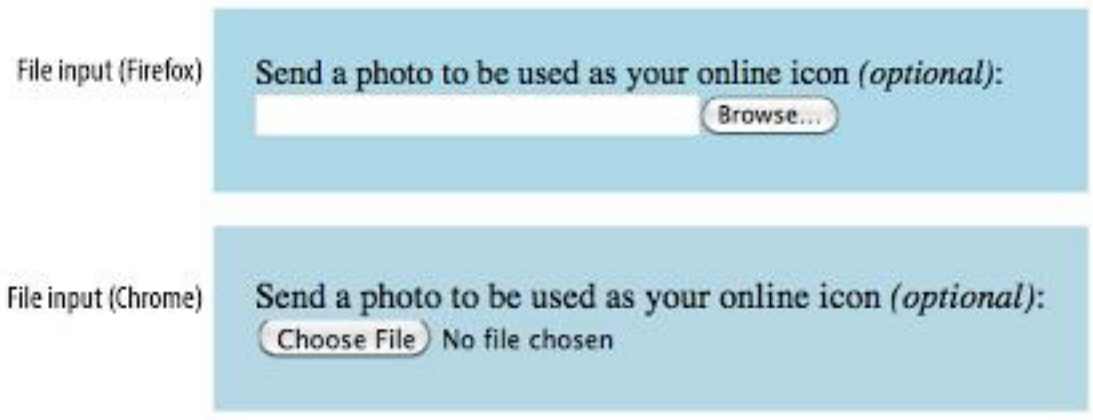


Figure 9-15. A file selection form field.

Hidden controls

`<input type="hidden">`

Hidden control field

There may be times when you need to send information to the form processing application that does not come from the user. In these instances, you can use a hidden form control that sends data when the form is submitted, but is not visible when the form is displayed in a browser.

Hidden controls are added using the `input` element with the `type` set to `hidden`. Its sole purpose is to pass a name/value pair to the server when the form is submitted. In this example, a hidden form element is used to provide the location of the appropriate thank-you document to display when the transaction is complete.

```
<input type="hidden" name="success-link" value="http://www.example.com/
littlechair_thankyou.html">
```

WARNING

It is possible for users to access and manipulate hidden form controls. If you should become a professional web developer, you will learn to program defensively for this sort of thing.

I've worked with forms that have had dozens of hidden controls in the `form` element before getting to the parts that the user actually fills out. This is the kind of information you get from the application programmer, system administrator, or whoever is helping you get your forms processed. If you are using a canned script, be sure to check the accompanying instructions to see if any hidden form variables are required.

Obras protegidas por Direitos de Autor

Date and time controls (HTML5)

If you've ever booked a hotel or a flight online, you've no doubt used a little calendar widget for choosing the date. Chances are, that little calendar was created using JavaScript. HTML5 introduced six new input types that make date and time selection widgets part of a browser's standard built-in display capabilities (just as they can display checkboxes, pop-up menus, and other widgets today). The date and time pickers are implemented on only a few browsers as of this writing, such as Opera, shown in Figure 9-16, but on non-supporting browsers, the date and time input types display as a perfectly usable text entry field instead.

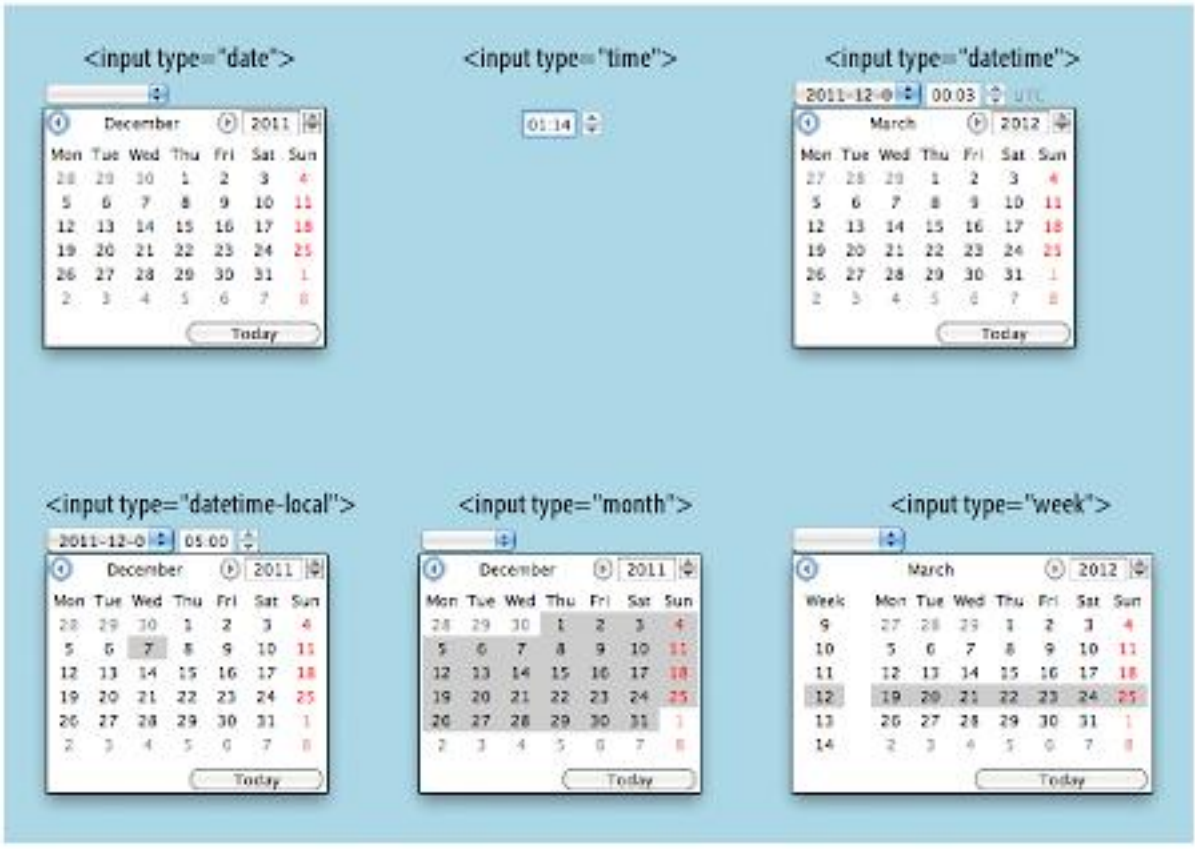


Figure 9-16. Date and time picker inputs (in Opera 11 on Mac OS X).

The new date- and time-related input types are as follows:

<input type="date" **name="***name***" value="***2004-01-14***">**

Creates a date input control, such as a pop-up calendar, for specifying a date (year, month, day). The initial value must be provided in ISO date format (**YYYY-MM-DD**).

<input type="time" **name="***name***" value="***03:13:00***">**

Creates a time input control for specifying a time (hour, minute, seconds, fractional sections) with no time zone indicated. The value is provided as **hh:mm:ss**.

<input type="date">

Date input control

NEW IN HTML5

<input type="time">

Time input control

NEW IN HTML5

<input type="datetime">

Date/time control with time zone

NEW IN HTML5

<input type="datetime-local">

Date/time control with no time zone

NEW IN HTML5

<input type="month">

Specifies a month in a year

NEW IN HTML5

<input type="week">

Specifies a particular week in a year

NEW IN HTML5

Obras protegidas por Direitos de Autor

image not available

Both the **number** and **range** input types accept the **min** and **max** attributes for specifying the minimum and maximum values allowed for the input (again, the browser could check that the user input complies with the constraint). Both **min** and **max** are optional, and you can also set one without the other.

The **step** attribute allows developers to specify the acceptable increments for numerical input. The default is 1. A value of .5 would permit values 1, 1.5, 2, 2.5, etc.; a value of 100 would permit 100, 200, 300, and so on. You can also set the **step** attribute to **any** to explicitly accept any value increment.

Again, browsers that don't support these new input types display a standard text input field instead, which is a fine fallback.

Color selector (HTML5)

`<input type="color">`

Color picker

NEW IN HTML5

The intent of the color control type is to create a pop-up color picker for visually selecting a color value similar to those used in operating systems or image-editing programs. Values are provided in hexadecimal RGB values (#RRGGBB). Figure 9-18 shows the color picker widget in Opera 11. Non-supporting browsers display the default text input instead.

```
<label>Your favorite color: <input type="color" name="favorite"></label>
```

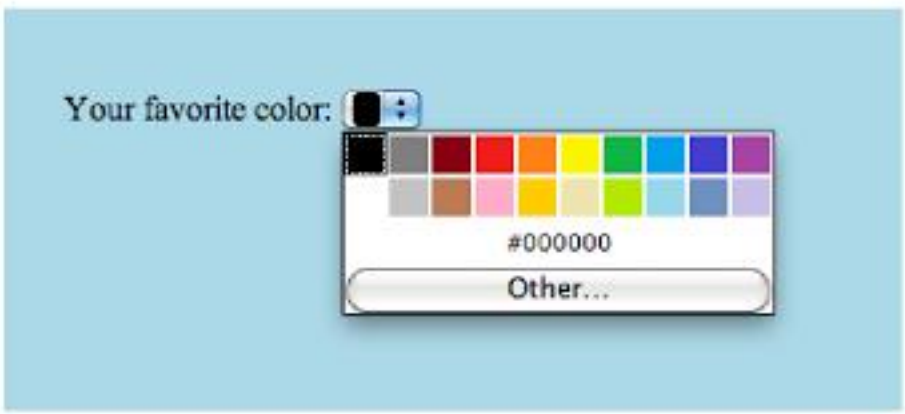


Figure 9-18. The `color` input type (in Opera 11 on Mac OS X).

That wraps up the form control roundup. Learning how to insert form controls is one part of the forms production process, but any web developer worth her salt will take the time to make sure the form is as accessible as possible. Fortunately, there are a few things we can do in markup to describe the form's structure.

Obras protegidas por Direitos de Autor

A Few More HTML5 Form Elements

For the sake of completeness, let's look at the remaining form elements that are new in HTML5. As of this writing, they are poorly supported, and are somewhat esoteric anyway, so you may wait a while to add these to your HTML toolbox. We've already covered the **datalist** element for providing suggested values for text inputs. HTML5 also introduced the following elements:

progress

`<progress>…</progress>`

Indicates the state of an ongoing process

NEW IN HTML5

The **progress** element gives users feedback on the state of an ongoing process, such as a file download. It can have a specific end value (provided with the **max** attribute) or just indicate that something is happening (such as waiting for a server to respond).

```
Percent downloaded: <progress max="100" name="fave">0</progress>
```

meter

`<meter>…</meter>`

Indicates the state of an ongoing process

NEW IN HTML5

meter is similar to **progress**, but it always represents a measurement within a known range of values (also known as a gauge). It has a number of attributes: **min** and **max** indicate the highest and lowest values for the range; **low** and **high** could be used to trigger warnings at undesirable levels; and **optimum** specifies a preferred value. The values would most likely be updated with JavaScript dynamically during the process.

```
<meter min="0" max="100" name="download">50%</meter>
```

output

`<output>…</output>`

Calculated output value

NEW IN HTML5

Simply put, the **output** element provides a way to indicate the results of a calculation by a script or program and associate it with inputs that affected the calculation.

keygen

`<keygen>`

Key pair generator

NEW IN HTML5

The **keygen** element represents a control for making a key pair (used to ensure privacy). When the form is submitted, the private key is stored locally, and the public key is packaged and sent to the server. Don't worry; I'm a little foggy on what this all means, too. You can read about public-key cryptography (*en.wikipedia.org/wiki/Public-key_cryptography*) and explain it to me when you figure it out.

Obras protegidas por Direitos de Autor

Form Accessibility Features

It is essential to consider how users without the benefit of visual browsers will be able to understand and navigate through your web forms. The **label**, **fieldset**, and **legend** form elements improve accessibility by making the semantic connections between the components of a form clear. The resulting markup is not only more semantically rich, but there are also more elements available to act as "hooks" for style sheet rules. Everybody wins!

Labels

Although we may see the label "Address" right next to a text field for entering an address in a visual browser, in the source, the label and field input may be separated. The **label** element associates descriptive text with its respective form field. This provides important context for users with speech-based browsers.

Each **label** element is associated with exactly one form control. There are two ways to use it. One method, called implicit association, nests the control and its description within a **label** element. In the following example, **labels** are assigned to individual checkboxes and their related text descriptions. (By the way, this is the way to label radio buttons and checkboxes. You can't assign a label to the entire group.)

```
<ul>
  <li><label><input type="checkbox" name="genre" value="punk"> Punk
rock</label></li>
  <li><label><input type="checkbox" name="genre" value="indie"> Indie
rock</label></li>
  <li><label><input type="checkbox" name="genre" value="hiphop"> Hip
Hop</label></li>
  <li><label><input type="checkbox" name="genre" value="rockabilly">
Rockabilly</label></li>
</ul>
```

The other method, called explicit association, matches the label with the control's **id** reference. The **for** attribute says which control the label is for. This approach is useful when the control is not directly next to its descriptive text in the source. It also offers the potential advantage of keeping the label and the control as two distinct elements, which may come in handy when aligning them with style sheets.

```
<label for="form-login-username">Login account</label>
<input type="text" name="login" id="form-login-username">

<label for="form-login-password">Password</label>
<input type="password" name="password" id="form-login-password">
```

Another advantage to using labels is that users can click or tap anywhere on them to select the form control. Users with touch devices will appreciate the larger tap target.

WARNING

iOS devices as of this writing do not make implicit labels clickable, so that behavior needs to be created with JavaScript. I know we haven't done any JavaScript yet, but if you are wondering, the fix looks like this:

```
document.getElementsByTagName
('label').setAttribute
('onclick','');
```

TIP

To keep form-related **id**s distinct from other **id**s on the page, consider prefacing them with "form-" as shown in the examples.

Another technique for keeping forms organized is to give the form element an ID name and include it as a prefix in the IDs for the controls it contains as follows:

```
<form id="form-login">

<input id="form-login-
username">

<input id="form-login-
password">
```


Obras protegidas por Direitos de Autor

WARNING

Fieldsets and legends tend to throw some curveballs when it comes to styling. For example, background colors in fieldsets are handled differently from browser to browser. Legends are unique in that their text doesn't wrap. The solution is to put a `span` *or* `b` *element in them and control presentation of the contained element without sacrificing accessibility. Be sure to do lots of testing if you style these form elements.*

fieldset and legend

The **fieldset** element indicates a logical group of form controls. A **fieldset** may also include a **legend** element that provides a caption for the enclosed fields.

Figure 9-19 shows the default rendering of the following example, but you could use style sheets to change the way the **fieldset** and **legend** appear.

```
<fieldset>
  <legend>Mailing List Sign-up</legend>
  <ul>
    <li><label>Add me to your mailing list <input type="radio"
    name="list" value="yes" checked="checked"></label></li>
    <li><label>No thanks <input type="radio" name="list" value="no">
  </label></li>
  </ul>
</fieldset>

<fieldset>
  <legend>Customer Information</legend>
  <ol>
    <li><label>Full name: <input type="text" name="username"></label></
li>
    <li><label>Email: <input type="text" name="email"></label></li>
    <li><label>State: <input type="text" name="state"></label></li>
  </ol>
</fieldset>
```

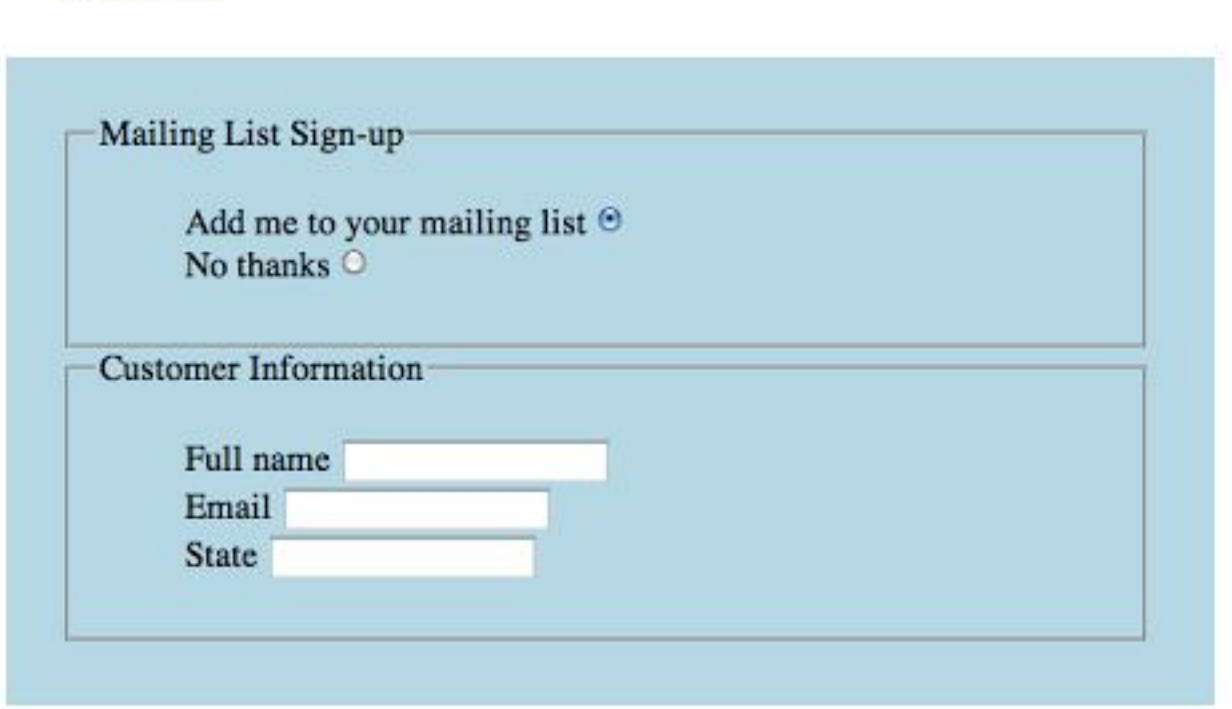


Figure 9-19. The default rendering of fieldsets and legends.

Obras protegidas por Direitos de Autor

exercise 9-4 | labels and fieldsets

Our contest form is working, but we need to label it appropriately and create some **fieldset**s to make it more usable on assistive devices. Once again, open the *contest_entry.html* document and follow these steps.

I like to start with the broad strokes and fill in details later, so we'll begin this exercise by organizing the form controls into fieldsets, and then we'll do all the labeling. You could do it the other way around, and ideally, you'd just mark up the labels and fieldsets as you go along instead of adding them all later.

1. The "Contest Entry Information" at the top of the form is definitely conceptually related, so let's wrap it all in a **fieldset** element. Change the markup of the section title from a paragraph (**p**) to a **legend** for the fieldset.

```
<fieldset>
  <legend>Contest Entry Information</legend>
  <ul>
    <li>Name: <input type="text"
name="username"></li>
    …
  </ul>
</fieldset>
```

2. Next, group the Color, Features, and Size questions in a big fieldset with the legend "Custom Shoe Design" (the text is there; you just need to change it from a **p** to a **legend**).

```
<h2>Design your custom Forcefields:</h2>
<fieldset>
<legend>Custom Shoe Design</legend>
  Color…
  Features…
  Size…
</fieldset>
```

3. Create another fieldset just for the Color options, again changing the description in a paragraph to a **legend**. Do the same for the Features and Size sections. In the end, you will have three fieldsets contained within the larger "Custom Shoe Design" fieldset. When you are done, save your document and open it in a browser. It should now look very close to the final form shown in Figure 9-2, given the expected browser differences.

```
<fieldset>
<legend>Color <em>(choose one)</em>:</legend>
   <ul>…</ul>
</fieldset>
```

4. OK, now let's get some labels in there. In the Contest Entry Information fieldset, explicitly tie the label to the text input using the **for/id** label method. I've done the first one for you; you do the other three.

```
<li><label for="form-name">Name:</label> <input
type="text" name="username" id="form-name"></li>
```

5. For the radio and checkbox buttons, wrap the **label** element around the **input** and its value label. In this way, the button will be selected when the user clicks or taps anywhere inside the label element. Here is the first one; you do the other seven.

```
<li><label><input type="radio" name="color"
value="red"> Red</label></li>
```

Save your document, and you're done! Labels don't have any effect on how the form looks by default, but you can feel good about the added semantic value you've added and maybe even use them to apply styles at another time.

Form Layout and Design

I can't close this chapter without saying a few words about form design, even though this chapter is about markup, not presentation.

Usable forms

A poorly designed form can ruin a user's experience on your site and negatively impact your business goals. Badly designed forms mean lost customers, so it is critical to get it right—both on the desktop and for small-screen devices with their special requirements. You want the path to a purchase or other action to be as frictionless as possible.

Obras protegidas por Direitos de Autor

The topic of good web form design is a rich one that could fill a book in itself. In fact, there is such a book: *Web Form Design* (Rosenfeld Media) by web form expert Luke Wroblewski, and I recommend it highly. Luke's subsequent book, *Mobile First* (A Book Apart), includes tips for how to format forms in a mobile context. You can browse over a hundred articles about forms on his site here: *www.lukew.com/ff?tag=forms*.

Here I'll offer just a very small sampling of tips from *Web Form Design* to get you started, but the whole book is worth a read.

Avoid unnecessary questions.

Help your users get through your form as easily as possible by not including questions that are not absolutely necessary to the task at hand. Extra questions, in addition to slowing things down, may make a user wary of your motivations for asking. If you have another way of getting the information (for example, the type of credit card can be determined from the first four numbers of the account), then use alternative means and don't put the burden on the user. If there is information that might be nice to have but is not required, consider asking at a later time, after the form has been submitted and you have built a relationship with the user.

Consider impact of label placement.

The position of the label relative to the input affects the time it takes to fill out the form. The less the user's eye needs to bounce around the page, the quicker the form completion. Putting the labels above their respective fields creates a single alignment for faster scans and completion, particularly when asking for familiar information (username, address, etc.). Top-positioned labels can also accommodate labels of varying lengths and work best on narrow, small-screen devices. They do result in a longer form, however, so if vertical space is a concern, you can position the labels to the left of the inputs. Left alignment of labels results in the slowest form completion, but it may be appropriate if you want the user to slow down or be able to scan and consider the types of information required in the form.

Choose input types carefully.

As you've seen in this chapter, there are quite a few input types to choose from, and sometimes it's not easy to decide which one to use. For example, a list of options could be presented as a pull-down menu or a number of choices with checkboxes. Weigh the pros and cons of each control type carefully, and follow up with user testing.

Group related inputs.

It is easier to parse the many fields, menus, and buttons in a form if they are visually grouped by related topic. For example, a user's contact information could be presented in a compact group so that five or six inputs are perceived as one unit. Usually, all you need is a very subtle indication, such as a fine horizontal rule and some extra space. Don't overdo it.

Obras protegidas por Direitos de Autor

Clarify primary and secondary actions.

The primary action at the end of the form is usually some form of submit button ("Buy," "Register," etc.) that signals the completion of the form and the readiness to move forward. You want that button to be visually dominant and easy to find (aligning it along the main axis of the form is helpful as well). Secondary actions tend to take you a step back, such as clearing or resetting the form. If you must include a secondary action, make sure that it is styled to look different and less important than the primary action. It is also a good idea to provide an opportunity to undo the action.

Styling Forms

As we've seen in this chapter, the default rendering of form markup is not up to par with the quality we see on most professional web forms today. As for other elements, you can use style sheets to create a clean form layout as well as change the appearance of most form controls. Something as simple as nice alignment and a look that is consistent with the rest of your site can go a long way toward improving the impression you make on a user.

Keep in mind that form widgets are drawn by the browser and are informed by operating system conventions. However, you can still apply dimensions, margins, fonts, colors, borders, and background effects to form elements such as text inputs, select menus, textareas, fieldsets, labels, and legends. Just be sure to test in a variety of browsers to check for unpleasant surprises. Chapter 18, CSS Techniques, in Part III, lists some specific techniques once you have more experience with CSS. For more help, a web search for "CSS for forms" will turn up a number of tutorials.

Test Yourself

Ready to put your web form know-how to the test? Here are a few questions to make sure you've gotten the basics.

1. Decide whether each of these forms should be sent via the GET or POST method:
   a. A form for accessing your bank account online _____
   b. A form for sending t-shirt artwork to the printer _____
   c. A form for searching archived articles _____
   d. A form for collecting long essay entries _____

Obras protegidas por Direitos de Autor

2. Which form control element is best suited for the following tasks? When the answer is "input," be sure to also include the type. Some tasks may have more than one correct answer.

   a. Choose your astrological sign from 12 signs.

   b. Indicate whether you have a history of heart disease (yes or no).

   c. Write up a book review.

   d. Select your favorite ice cream flavors from a list of eight flavors.

   e. Select your favorite ice cream flavors from a list of 25 flavors.

3. Each of these markup examples contains an error. Can you spot what it is?

   a. `<input name="country" value="Your country here.">`

   b. `<checkbox name="color" value="teal">`

   c.
```
<select name="popsicle">
  <option value="orange">
  <option value="grape">
  <option value="cherry">
</select>
```

   d. `<input type="password">`

   e. `<textarea name="essay" height="6" width="100">Your story.</textarea>`

Element Review: Forms

We covered this impressive list of elements and attributes related to forms in this chapter. Elements marked with (HTML5) are new in the HTML5 specification.

Element and attributes	Description
`button`	Generic input button
`name="text"`	Supplies a unique variable name for the control
`type="submit|reset|button"`	The type of custom button
`value="text"`	Specifies the value to be sent to the server
`datalist` *[HTML5]*	Provides a list of options for text inputs
`fieldset`	Groups related controls and labels
`form`	Form element
`action="url"`	Location of forms processing program (*required*)
`method="get|post"`	The method used to submit the form data
`enctype="content type"`	The encoding method, generally either **`application/x-www-form-urlencoded`** (default) or **`multipart/form-data`**

Obras protegidas por Direitos de Autor

Element and attributes	Description
`input`	Creates a variety of controls, based on the **`type`** value
`autofocus`	Indicates the control should be ready for input when the document loads
`type="submit|reset|button|text|password|checkbox|radio|image|file|hidden|email|tel|search|url|date|time|datetime|datetime-local|month|week|number|range|color "`	The type of input
`disabled`	Disables the input so it cannot be selected
`form="form id value"`	Associates the control with a specified form
See Table 9-1 for a full list of attributes associated with each input type.	
`keygen` *[HTML5]*	Generates key pairs for secure transaction certificates
`autofocus`	Indicates the control should be highlighted and ready for input when the document loads
`challenge="challenge string"`	Provides a challenge string to be submitted with the key
`disabled`	Disables the control so it cannot be selected
`form="form id value"`	Associates the control with a specified form
`keytype="keyword"`	Identifies the type of key to be generated (e.g., **`rsa`** or **`ec`**)
`name="text"`	Gives control an identifying name
`label`	Attaches information to controls
`for="text"`	Identifies the associated control by its **`id`** reference
`form="form id value"`	Associates the control with a specified form
`legend`	Assigns a caption to a **`fieldset`**
`meter` *[HTML5]*	Represents a fractional value within a known range
`form="form id value"`	Associates the control with a specified form
`high="number"`	Indicates the range that is considered "high" for the gauge
`low="number"`	Indicates the range that is considered "low" for the gauge
`max="number"`	Specifies the highest value for the range
`min="number"`	Specifies the lowest value for the range
`optimum="number"`	Indicates the number considered to be "optimum"
`value="number"`	Specifies the actual or measured value
`optgroup`	Defines a group of options
`disabled`	Disables the **`optgroup`** so it cannot be selected
`label="text"`	Supplies label for a group of options
`option`	An option within a select menu control
`disabled`	Disables the **`option`** so it cannot be selected
`label="text"`	Supplies an alternate label for the option
`selected`	Preselects the option
`value="text"`	Supplies an alternate value for the option

Obras protegidas por Direitos de Autor

Element and attributes	Description
`output` *[HTML5]*	Represents the results of a calculation
`for="text"`	Creates relationship between output and another element
`form="form id value"`	Associates the control with a specified form
`name="text"`	Supplies a unique variable name for the control
`progress` *[HTML5]*	Represents the completion progress of a task (can be used even if the maximum value of the task is not known)
`form="form id value"`	Associates the control with a specified form
`max="number"`	Specifies the total value or final size of the task
`value="number"`	Specifies how much of the task has been completed
`select`	Pull-down menu or scrolling list
`autofocus`	Indicates the control should be highlighted and ready for input when the document loads
`disabled`	Indicates the control is nonfunctional; can be activated with a script
`form="form id value"`	Associates the control with a specified form
`multiple`	Allows multiple selections in a scrolling list
`name="text"`	Supplies a unique variable name for the control
`readonly`	Makes the control unalterable by the user
`required`	Indicates the user input is required for this control
`size="number"`	The height of the scrolling list in text lines
`textarea`	Multiline text entry field
`autofocus`	Indicates the control should be highlighted and ready for input when the document loads
`cols="number"`	The width of the text area in characters
`dirname="text"`	Allows text directionality to be specified
`disabled`	Disables the control so it cannot be selected
`form="form id value"`	Associates the control with a specified form
`maxlength="text"`	Specifies the maximum number of characters the user can enter
`name="text"`	Supplies a unique variable name for the control
`placeholder="text"`	Provides a short hint to help the user enter the correct data
`readonly`	Makes the control unalterable by the user
`required`	Indicates user input is required for this control
`rows="number"`	The height of the text area in text lines
`wrap="hard|soft"`	Controls whether line breaks in the text input are returned in the data; **`hard`** preserves line breaks, while **`soft`** does not

Obras protegidas por Direitos de Autor

Table 9-1. Available attributes for each input type

	submit	reset	button	text	password	checkbox	radio	image	file	hidden
accept									•	
alt								•		
checked						•	•			
disabled	•	•	•	•	•	•	•	•	•	•
maxlength				•	•				•	
name	•	•	•	•	•	•	•	•	•	•
readonly				•	•	•	•		•	
size				•	•				•	
src								•		
value	•	•	•	•	•	•	•		•	•
HTML5-only										
autocomplete				•	•					
autofocus	•	•	•	•	•	•	•	•	•	
form	•	•	•	•	•	•	•	•	•	•
formaction	•							•		
formenctype	•							•		
formmethod	•							•		
formnovalidate	•							•		
formtarget	•							•		
height								•		
list				•						
max										
min										
multiple									•	
pattern				•	•					
placeholder				•	•					
required				•	•	•	•		•	
step										
width								•		

Obras protegidas por Direitos de Autor

	email	telephone, search, url	number	range	date, time, datetime, datetime-local, month, week	color
accept						
alt						
checked						
disabled	•	•	•	•	•	•
maxlength	•	•				
name	•	•	•	•	•	•
readonly	•	•	•		•	
size	•	•				
src						
value	•	•	•	•	•	•
HTML5-only						
autocomplete	•	•	•	•	•	•
autofocus	•	•	•	•	•	•
form	•	•	•	•	•	•
formaction						
formenctype						
formmethod						
formnovalidate						
formtarget						
height						
list	•	•	•	•	•	•
max			•	•	•	
min			•	•	•	
multiple	•					
pattern	•	•				
placeholder	•	•				
required	•	•	•		•	
step			•	•	•	
width						

Obras protegidas por Direitos de Autor

CHAPTER 10

WHAT'S UP, HTML5?

We've been using HTML5 elements in the past several chapters, but there is a lot more to the HTML5 specification than new markup possibilities (although that is an important part). HTML5 is actually a bundle of new methods for accomplishing tasks that previously required special programming or proprietary plug-in technology such as Flash or Silverlight. It offers a standardized, open source way to put audio, video, and interactive elements on the page as well as the ability to store data locally, work offline, take advantage of location information, and more. With HTML5 for common tasks, developers can rely on built-in browser capabilities and not need to reinvent the wheel for every application.

IN THIS CHAPTER

What HTML5 is and *isn't*

A brief history of HTML

New elements and attributes

HTML5 APIs

Adding video and audio

The canvas element

HTML5 offers so many promising possibilities, in fact, that it has become something of a buzzword with connotations far beyond the spec itself. When marketers and journalists use the term "HTML5," they are sometimes referring to CSS3 techniques or any new web technology that isn't Flash. In this chapter you'll learn what is actually included in the spec, and you can join the rest of us in being slightly irked when the HTML5 label is applied incorrectly. The important thing, however, is that mainstream awareness of web standards is certainly a win and makes our job easier when communicating with clients.

Of course, with any spec in development, browser support is uneven at best. There are some features that can be used right away and some that aren't quite ready for prime time. But this time around, instead of waiting for the entire spec to be "done," browsers are implementing one feature at a time, and developers are encouraged to begin using them (see the "Tracking Browser Support" sidebar). I should also mention that the HTML5 spec is evolving rapidly and parts are likely to have changed by the time you are reading this. I'll do my best to give you a good overview, and you can decide which features to research and follow up on your own.

Much of what's new in HTML5 requires advanced web development skills, so it is unlikely you'll use them right away (if ever), but as always, I think it is beneficial to everyone to have a basic familiarity with what can be done. And

Obras protegidas por Direitos de Autor

"basic familiarity" is what I'm aiming at with this chapter. For more in-depth discussions of HTML5 features, I recommend the following books:

- *HTML5: Up and Running* by Mark Pilgrim (O'Reilly and Google Press)
- *Introducing HTML5* by Bruce Lawson and Remy Sharp (New Riders)

I feel it's only fair to warn you that this chapter is the cod liver oil of this book. Not pleasant to get down, but good for you. An understanding of the big picture and the context of why we do things the way we do is something any budding web designer should have.

Tracking Browser Support

There are several nice resources out there to help you know which HTML5 features are ready to use. Most show support for CSS properties and selectors as well.

- When Can I Use… (*caniuse.com*)
- HTML5 Please (*html5please.com*)
- "Comparison of Layout Engines (HTML5)" on Wikipedia (*en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML_5)*)

A Funny Thing Happened on the Way to XHTML 2

Understanding where we've been provides useful context for where we are going, so let's kick this off with a quick history lesson. We'll start at the very beginning.

A "don't blink or you'll miss it" history of HTML

The story of HTML, from Tim Berners-Lee's initial draft in 1991 to the HTML5 standard in development today, is both fascinating and tumultuous. Early versions of HTML (HTML+ in 1994 and HTML 2.0 in 1995) built on Tim's early work with the intent of making it a viable publishing option. But when the World Wide Web (as it was adorably called back in the day) took the world by storm, browser developers, most notably Mosaic Netscape and later Microsoft Internet Explorer, didn't wait for any stinkin' standards. They gave the people what they wanted by creating a slew of browser-specific elements for improving the look of pages on their respective browsers. This divisive one-upping is what has come to be known as the Browser Wars. As a result, it became common in the late 1990s to create two separate versions of a site that targeted each of the Big Two browsers.

In 1996, the newly formed W3C put a stake in the ground and released its first Recommendation: HTML 3.2. It is a snapshot of all the HTML elements in common use at the time, and includes many presentational extensions to HTML that were the result of the Netscape/IE feud and the lack of a style sheet alternative. HTML 4.0 (1998) and HTML 4.01 (the slight revision that superseded it in 1999) aimed to get HTML back on track by emphasizing the separation of structure and presentation and improving accessibility. All matters of presentation were handed over to the newly minted Cascading Style Sheets standard that was gaining support.

NOTE

For a detailed history of the beginnings of the World Wide Web and HTML, read David Raggett's account from his book Raggett on HTML4 (Addison-Wesley), available on the W3C site (www.w3.org/People/Raggett/book4/ch02.html).

Obras protegidas por Direitos de Autor

Enter XHTML

Around the same time that HTML 4.01 was in development, folks at the W3C became aware that one limited markup language wasn't going to cut it for describing all the sorts of information (chemical notation, mathematical equations, multimedia presentations, financial information, and so on) that might be shared over the Web. Their solution: XML (eXtensible Markup Language), a metalanguage for creating markup languages. XML was a simplification of SGML (Standardized Generalized Markup Language), the big kahuna of metalanguages that Tim Berners-Lee used to create his original HTML application. But SGML itself proved to be more complex than the Web required.

The W3C had a vision of an XML-based Web with many specialized markup languages working together—even within a single document. Of course, to pull that off, everyone would have to mark up documents very carefully, strictly abiding by XML syntax, to rule out potential confusion.

Their first step was to rewrite HTML according to the rules of XML so that it could play well with others. The result is XHTML (eXtensible HTML). The first version, XHTML 1.0, is nearly identical to HTML 4.01, sharing the same elements and attributes, but with stricter requirements for how markup must be done (see the "XHTML Markup Requirements" sidebar).

HTML 4.01, along with XHTML 1.0, its stricter XML-based sibling, became the cornerstone of the web standards movement (see the sidebar "The Web Standards Project"). They are still the most thoroughly and consistently supported standards as of this writing (although HTML5 is quickly gaining steam).

But the W3C didn't stop there. With a vision of an XML-based Web in mind, they began work on XHTML 2.0, an even bolder attempt to make things work "right" than HTML 4.01 had been. The problem was that it was not backward-compatible with old standards and browser behavior. The writing and approval process dragged on for years with no browser implementation. Without browser implementation, XHTML 2.0 was stuck.

XHTML Markup Requirements

- Element and attribute names must be lowercase. In HTML, element and attribute names are not case-sensitive.
- All elements must be closed (terminated). Empty elements are closed by adding a slash before the closing bracket (for example, `<br/>`).
- Attribute values must be in quotation marks. Single or double quotation marks are acceptable as long as they are used consistently. Furthermore, there should be no extra whitespace (character spaces or line returns) before or after the attribute value inside the quotation marks.
- All attributes must have explicit attribute values. XML (and therefore XHTML) does not support attribute minimization, the SGML practice in which certain attributes can be reduced to just the attribute value. So, while in HTML you can write `checked` to indicate that a form button be checked when the form loads, in XHTML you need to explicitly write out `checked="checked"`.
- Proper nesting of elements is strictly enforced. Some elements have new nesting restrictions.
- Special characters must always be represented by character entities (e.g., `&` for the & symbol).
- Use `id` instead of `name` as an identifier.
- Scripts must be contained in a CDATA section so they will be treated as simple text characters and not parsed as XML markup. Here is an example of the syntax:

```
<script type="type/javascript">
  // <![CDATA[
  ... JavaScript goes here...
  // ]]>
</script>
```

The Web Standards Project

In 1998, at the height of the browser wars, a grassroots coalition called the Web Standards Project (WaSP for short) began to put pressure on browser creators (primarily Netscape and Microsoft at the time) to start sticking to the open standards as documented by the W3C. Not stopping there, it educated the web developer community on the many benefits of developing with standards. Its efforts revolutionized the way sites are created and supported. Now browsers (even Microsoft) brag of standards support while continuing to innovate. You can read its mission statement, history, and descriptions of its current efforts on the WaSP site (*webstandards.org*).

Obras protegidas por Direitos de Autor

Hello HTML5!

HTML5 aims to make HTML more useful for creating web applications.

Meanwhile...

In 2004, members of Apple, Mozilla, and Opera formed the Web Hypertext Application Technology Working Group (WHATWG, *whatwg.org*), separate from the W3C. The goal of the WHATWG was to further the development of HTML to meet new demands in a way that was consistent with real-world authoring practices and browser behavior (in contrast to the start-from-scratch ideal that XHTML 2.0 described). Their initial documents, Web Applications 1.0 and Web Forms 1.0, were rolled together into HTML5, which is still in development under the guidance of an editor, Ian Hickson (currently of Google).

The W3C eventually established its own HTML5 Working Group (also led by Hickson) based on the work done by the WHATWG. As of this writing, work on the HTML5 specification is happening in both organizations in tandem, sometimes with conflicting results. It is not yet a formal Recommendation as of this writing, but that isn't stopping browsers from implementing it a little at a time.

NOTE

The WHATWG maintains what it calls the HTML "Living Standard" (meaning the group isn't giving it a version number) at www.whatwg.org. It is nearly identical to HTML5, but it includes a few extra elements and attributes that the W3C isn't quite ready to adopt, and it has a slightly different lineup of APIs.

And XHTML 2.0? At the end of 2009, the W3C officially put it out of its misery, pulling the plug on the working group and putting its resources and efforts into HTML5.

So that's how we got here, and it's a whole lot of prelude to the meat of this chapter, which of course is the new features that HTML5 offers. I also encourage you to read the sidebar "HTML5 Fun Facts" for more juicy information on the specification itself. In this section, I'll introduce what's new in HTML5, including:

- A new DOCTYPE
- New elements and attributes
- Obsolete 4.01 elements
- APIs

Obras protegidas por Direitos de Autor

HTML5 Fun Facts

HTML5 both builds on previous versions of HTML and introduces some significant departures. Here are some interesting tidbits about the HTML5 specification itself.

- HTML5 is based on HTML 4.01 Strict, the version of HTML that did not include any presentation-based or other deprecated elements and attributes. That means the vast majority of HTML5 is made up of the same elements we've been using for years, and browsers know what to do with them.
- HTML5 does not use a DTD (Document Type Definition), which is a document that defines all of the elements and attributes in a markup language. It is the way you document a language in SGML, and if you'll remember, HTML was originally crafted according to the rules of SGML. HTML 4.01 was defined by three separate DTDs: Transitional (including legacy elements that were marked as "deprecated," or soon to be obsolete), Strict (deprecated features stripped out, as noted earlier), and Frameset (for documents broken into individually scrolling frames, a technique that is now considered obsolete).
- HTML5 is the first HTML specification that includes detailed instructions for how browsers should handle malformed and legacy markup. It bases the instructions on legacy browser behavior, but for once, there is a standard protocol for browser makers to follow when browsers encounter incorrect or non-standard markup.
- HTML5 can also be written according to the stricter syntax of XML (called the XML serialization of HTML5). Some developers have come to prefer the tidiness of well-formed XHTML (lowercase element names, quoted attribute values, closing all elements, and so on), so that way of writing is still an option, although not required. In edge cases, an HTML5 document may be required to be served as XML in order to work with other XML applications, in which case it can use the XML syntax and be ready to go.
- In addition to markup, HTML5 defines a number of APIs (Application Programming Interfaces). APIs make it easier to communicate with web-based applications. They also move some common processes (such as audio and video players) into native browser functionality.

In the Markup Department

We'll start with a look at the markup aspects of HTML5, and then we'll move on to the APIs.

A minimal DOCTYPE

As we saw in Chapter 4, HTML documents should begin with a Document Type Declaration (DOCTYPE declaration) that identifies which version of HTML the document follows. The HTML5 declaration is short and sweet:

```
<! DOCTYPE html>
```

Compare that to a declaration for a Strict HTML 4.01 document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/HTML4.01/strict.dtd">
```

Why so complicated? In HTML 4.01 and XHTML 1.0 and 1.1, the declaration must point to the public DTD (Document Type Definition), a document that defines all of the elements in a markup language as well as the rules for using them. HTML 4.01 was defined by three separate DTDs: Transitional (including legacy elements such as font and attributes such as **align** that were marked as "deprecated," or soon to be obsolete), Strict (deprecated features stripped out), and Frameset (for documents broken into individually scrolling frames, a technique that is now considered obsolete). HTML5 does not have a DTD, which is why we have the simple DOCTYPE declaration. DTDs

Obras protegidas por Direitos de Autor

are a remnant of SGML and proved to be less helpful on the Web than originally thought, so the authors of HTML5 simply didn't use one.

Validators—software that checks that all the markup in a document is correct (see note)—use the DOCTYPE declaration to make sure the document abides by the rules of the specification it claims to follow. The sidebar "HTML DOCTYPES" lists all declarations in common use, should you need to write documents in HTML 4.01 or XHTML 1.0.

NOTE

To check whether your HTML document is valid, use the online validator at the W3C (validator.w3.org). An HTML5-specific validator is also available at html5.validator.nu. There is also a validator built into Adobe Dreamweaver that allows you to check your document against various specs as you work.

HTML DOCTYPES

The following lists all of the DOCTYPE declarations in common use.

HTML5

```
<!DOCTYPE html>
```

HTML 4.01 Transitional

The Transitional DTD includes deprecated elements and attributes:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
```

HTML 4.01 Strict

The Strict DTD omits all deprecated elements and attributes:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4.01/strict.dtd">
```

HTML 4.01 Frameset

If your document contains frames—that is, it uses **frameset** instead of **body** for its content—then identify the Frameset DTD:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/HTML4.01/frameset.dtd">
```

XHTML 1.0 Strict

The same as HTML 4.01 Strict, but reformulated according to the syntax rules of XML:

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

XHTML 1.0 Transitional

The same as HTML 4.01 Transitional, but reformulated according to the syntax rules of XML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset

The same as HTML 4.01 Frameset, but reformulated according to the syntax rules of XML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```


Obras protegidas por Direitos de Autor

Elements and attributes

HTML5 introduced a number of new elements. You'll find them sprinkled throughout this book, but Table 10-1 lists them all in one place.

Table 10-1. New elements in HTML5

`article`	`datalist`	`header`	`output`	`source`
`aside`	`details`	`hgroup`	`progress`	`summary`
`audio`	`embed`	`keygen`	`rp`	`time`
`bdi`	`figcaption`	`mark`	`rt`	`track`
`canvas`	`figure`	`meter`	`ruby`	`video`
`command`	`footer`	`nav`	`section`	`wbr`

NOTE

For a detailed list of all the ways HTML5 differs from HTML 4.01, see the W3C official document at www.w3.org/TR/html5-diff/.

New form input types

We covered the new form input control types in Chapter 9, but here they are at a glance: **`color`**, **`date`**, **`datetime`**, **`datetime-local`**, **`email`**, **`month`**, **`number`**, **`range`**, **`search`**, **`tel`**, **`time`**, **`url`**, and **`week`**.

New global attributes

Global attributes are attributes that can be applied to any element. The number of global attributes was expanded in HTML5, and many of them are brand new (as noted in Table 10-2). The W3C is still adding and removing attributes as of this writing, so it's worth checking in with the spec for the latest (*dev.w3.org/html5/spec/global-attributes.html#global-attributes*).

Table 10-2. Global attributes in HTML5

Attribute	Values	Description
`accesskey`	Single text character	Assigns an access key (shortcut key command) to the link. Access keys are also used for form fields. Users may access the element by pressing Alt-<*key*> (PC) or Ctrl-<*key*> (Mac).
`aria-*`	One of the standardized state or property keywords in WAI-ARIA (*www.w3.org/TR/wai-aria/states_and_properties*)	WAI-ARIA (Accessibile Rich Internet Applications) defines a way to make web content and applications more accessible to users with assistive devices. HTML5 allows any of the ARIA properties and roles to be added to elements. For example, a **`div`** used for a pop-up menu could include the attribute **`aria-haspopup`** to make that property clear to a user without a visual browser. See also the related **`role`** global attribute.
`class`	Text string	Assigns one or more classification names to the element.
`contenteditable`	**`true`** | **`false`**	NEW IN HTML5 Indicates the user can edit the element. This attribute is already well supported in current browser versions.

Obras protegidas por Direitos de Autor

Table 10-2. Global attributes in HTML5

Attribute	Values	Description
`contextmenu`	`id` of the `menu` element	NEW IN HTML5 Specifies a context menu that applies to the element. The context menu must be requested by the user, for example, by a right-click.
`data-*`	Text string or numerical data	NEW IN HTML5 Enables authors to create custom data-related attributes (the "*" is a symbol that means "anything"), for example, `data-length`, `data-duration`, `data-speed`, etc. so that the data can be used by a custom application or scripts.
`dir`	`ltr` | `rtl`	Specifies the direction of the element ("left to right" or "right to left").
`draggable`	`true` | `false`	NEW IN HTML5 A `true` value indicates the element is draggable, meaning it can be moved by clicking and holding on it, then moving it to a new position in the window.
`dropzone`	`copy` | `link` | `move` | `s:text/plain` | `f:`*`file-type`* (for example, `f:image/jpg`)	NEW IN HTML5 Indicates the element can accept dragged and dropped text or file data. The values are a space-separated list that includes what type of data it accepts (`s:text/plain` for text strings; `f:`*`file-type`* for file types) and a keyword that indicates what to do with the dropped content: `copy` results in a copy of the dragged data; `move` moves it to the new location; and `link` results in a link to the original data.
`hidden`	No value for HTML documents In XHTML, set a value `hidden="hidden"`	NEW IN HTML5 Prevents the element and its descendants from being rendered in the user agent (browser). Any scripts or form controls in hidden sections will still execute, but they will not be presented to the user.
`id`	Text string (may not begin with an number)	Assigns a unique identifying name to the element.
`lang`	Two-letter language code (see *www.loc.gov/standards/iso639-2/php/code_list.php*)	Specifies the language for the element by its language code.
`role`	One of the standard role keywords in WAI-ARIA (see *www.w3.org/TR/wai-aria/roles*)	NEW IN HTML5 Assigns one of the standardized WAI-ARIA roles to an element to make its purpose clearer to users with disabilities. For example, a `div` with contents that will display as a pop-up menu on visual browsers could be marked with `role="menu"` for clarity on screen readers.
`spellcheck`	`true` | `false`	NEW IN HTML5 Indicates the element is to have its spelling and grammar checked.
`style`	Semicolon-separated list of style rules (`property: value` pairs)	Associates style information with an element. For example: `<h1 style="color: red; border: 1px solid">Heading</h1>`
`tabindex`	Number	Specifies the position of the current element in the tabbing order for the current document. The value must be between 0 and 32,767. It is used for tabbing through links on a page or fields in a form and is useful for assistive browsing devices. A value of –1 is allowable to remove elements from the tabbing flow and make them focusable only by JavaScript.
`title`	Text string	Provides a title or advisory information about the element, typically displayed as a tooltip.

Obras protegidas por Direitos de Autor

Obsolete HTML 4.01 Markup

HTML5 also declared a number of elements in HTML 4.01 to be "obsolete" because they are presentational, antiquated, or poorly supported (Table 10-3). If you use them, browsers will support them, but I strongly recommend leaving them in the dust.

Table 10-3. HTML 4 elements that are now obsolete in HTML5

`acronym`	`dir`	`noframes`
`applet`	`font`	`strike`
`basefont`	`frame`	`tt`
`big`	`frameset`	
`center`	`isindex`	

Are you still with me? I know, this stuff gets pretty dry. That's why I've included Figure 10-1. It has nothing at all to do with HTML5, but I thought we could all use a little breather before taking on APIs.

Figure 10-1. This adorable baby red panda has nothing to do with HTML5. (Photo by Tara Menne.)

Meet the APIs

HTML specifications prior to HTML5 included only documentation of the elements, attributes, and values permitted in the language. That's fine for simple text documents, but the creators of HTML5 had their minds set on

Obras protegidas por Direitos de Autor

making it easier to create web-based applications that require scripting and programming. For that reason, HTML5 also defines a number of new APIs for making it easier to communicate with an application.

An API (Application Programming Interface) is a documented set of commands, data names, and so on, that lets one software application communicate with another. For example, the developers of Twitter documented the names of each data type (users, tweets, timestamps, and so on) and the methods for accessing them in an API document (*dev.twitter.com/docs*) that lets other developers include Twitter feeds and elements in their programs. That is why there are so many Twitter programs and widgets available. Amazon.com also opens up its product data via an API. In fact, publishers of all ilks are recognizing the power of having their content available via an API. You could say that APIs are hot right now.

But let's bring it back to HTML5, which includes APIs for tasks that traditionally required proprietary plug-ins (like Flash) or custom programming. The idea is that if browsers offer those features natively—with standardized sets of hooks for accessing them—developers can do all sorts of nifty things and count on them working in all browsers, just as we count on the ability to embed an image on a page today. Of course, we have a way to go before there is ubiquitous support of these cutting-edge features, but we're getting there steadily. Some APIs have a markup component, such as embedding multimedia with the new HTML5 `video` and `audio` elements. Others happen entirely behind the scenes with JavaScript or server-side components, such as creating web applications that work even when there is no Internet connection (Offline Web Application API).

NOTE

For a list of all the APIs, see the article "HTML Landscape Overview" by Erik Wilde (dret.typepad.com/dretblog/html5-api-overview.html). The W3C lists all the documents they maintain, many of which are APIs, at www.w3.org/TR/tr-title-all.

The W3C and WHATWG are working on *lots and lots* of APIs for use with web applications, all in varying stages of completion and implementation. Most have their own specifications, separate from the HTML5 spec itself, but they are generally included under the wide HTML5 umbrella that covers web-based applications. HTML5 includes specifications for these APIs:

Media Player API

For controlling audio and video players embedded on a web page, used with the new `video` and `audio` elements. We will take a closer look at audio and video later in this chapter.

Session History API

Exposes the browser history for better control over the Back button.

Offline Web Application API

Makes it possible for a web application to work even when there is no Internet connection. It does it by including a manifest document that lists all of the files and resources that should be downloaded into the browser's cache in order for the application to work. When a connection is available, it checks to see whether any of the documents have changed, then updates those documents.

Obras protegidas por Direitos de Autor

Editing API

Provides a set of commands that could be used to create in-browser text editors, allowing users to insert and delete text; format text as bold, italic, or as a hypertext link; and more. In addition, there is a new `contenteditable` attribute that allows any content element to be editable right on the page.

Drag and Drop API

Adds the ability to drag a text selection or file to a target area on the page or another web page. The `draggable` attribute indicates the element can be selected and dragged. The `dropzone` attribute is used on the target area and defines what type of content it can accept (text or file type) and what to do with it when it gets there (`copy`, `link`, `move`).

The following are just a handful of the APIs in development at the W3C with specifications of their own (outside HTML5):

Canvas API

The `canvas` element adds a dynamic, two-dimensional drawing space to a page. We'll take a look at it at the end of this chapter.

Web Storage API

Allows data to be stored in the browser's cache so that an application can use it later. Traditionally, that has been done with "cookies," but the Web Storage API allows more data to be stored. It also controls whether the data is limited to one session (`sessionStorage`: when the window is closed, the data is cleared) or based on domain (`localStorage`: all open windows pointed to that domain have access to the data).

Geolocation API

Lets users share their geographical location (longitude and latitude) so that it is accessible to scripts in a web application. This allows the app to provide location-aware features such as suggesting a nearby restaurant or finding other users in your area.

Web Workers API

Provides a way to run computationally complicated scripts in the background. This allows the browser to keep the web page interface quick and responsive to user actions while working on processor-intensive scripts at the same time. The Web Workers API is part of the HTML5 spec at the WHATWG, but at the W3C, it's been moved into a separate document.

Web Sockets API

Creates a "socket," which is an open connection between the browser client and the server. This allows information to flow between the client and the server in real time, with no lags for the traditional HTTP requests. It is useful for multiplayer games, chat, or data streams that update constantly, such as sports or stock tickers or social media streams.

NOTE

You can think of a web socket as an ongoing telephone call between the browser and server compared to the walkie-talkie, one-at-a-time style of traditional browser/server communication. (A hat tip to Jen Simmons for this analogy.)

Obras protegidas por Direitos de Autor

Some APIs have correlating HTML elements, such as the **audio** and **video** elements for embedding media players on a page, and the **canvas** element for adding a dynamic drawing area. In the following sections, we'll take a brief look at how those elements are put to use.

Video and Audio

In the earliest days of the World Wide Web (I know, I was there), it was possible to add a MIDI file to a web page for a little beep-boopy soundtrack (think early video games). It wasn't long before better options came along, including RealMedia and Windows Media, that allowed all sorts of audio and video formats to be embedded in a web page. In the end, Flash became the *de facto* embedded multimedia player thanks in part to its use by YouTube and similar video services.

What all of these technologies have in common is that they require third-party, proprietary plug-ins to be downloaded and installed in order to play the media files. Until recently, browsers did not have built-in capabilities for handling sound or video, so the plug-ins filled in the gap. With the development of the Web as an open standards platform, it seemed like time to make multimedia support part of browsers' out-of-the-box capabilities. Enter the new **audio** and **video** elements and their respective APIs.

Farewell Flash?

Apple's announcement that it would not support Flash on its iOS devices, *ever*, gave HTML5 an enormous push forward and eventually led to Adobe stopping development on its mobile Flash products. Not long after, Microsoft announced that it was discontinuing its Silverlight media player in lieu of HTML5 alternatives. As of this writing, HTML5 is a long way from being able to reproduce the vast features and functionality of Flash, but it's getting there gradually. That means we are likely to see Flash and Silverlight players on the desktop for years to come, but the trajectory away from plug-ins and toward web standards technologies seems clear.

The good news and the bad news

The good news is that the **audio** and **video** elements are well supported in modern browsers, including IE 9+, Safari 3+, Chrome, Opera, and Firefox 3.5+ for the desktop and iOS Safari 4+, Android 2.3+, and Opera Mobile (however, not Opera Mini).

But lest you envision a perfect world where all browsers are supporting audio and video in perfect harmony, I am afraid that it is not that simple. Although they have all lined up on the markup and JavaScript for embedding and controlling media players, unfortunately they have not agreed on which formats to support. Let's take a brief journey through the land of media file formats. If you want to add audio or video to your page, this stuff is important to understand.

How media formats work

When you prepare audio or video content for web delivery, there are two format decisions to make. The first is how the media is encoded (the algorithms used to convert the source to 1s and 0s and how they are compressed). The method used for encoding is called the codec, which is short for "code/decode" or "compress/decompress." There are a bazillion codecs out there (that's an estimate). Some probably sound familiar, like MP3; others might

Obras protegidas por Direitos de Autor

sound new, such as H.264, Vorbis, Theora, VP8, and AAC. Fortunately, only a few are appropriate for the Web, and we'll review them in a moment.

Second, you need to choose the container format for the media...you can think of it as a ZIP file that holds the compressed media and its metadata together in a package. Usually a container format can hold more than one codec type, and the full story is complicated. Because space is limited in this chapter, I'm going to cut to the chase and introduce the most common container/codec combinations for the Web. If you are going to add video or audio to your site, I encourage you to get more familiar with all of these formats. The books in the "For Further Reading: HTML5 Media" sidebar are a great first step.

Meet the video formats

For video, the most common options are:

- **Ogg container + Theora video codec + Vorbis audio codec.** This is typically called "Ogg Theora," and the file should have a *.ogv* suffix. All of the codecs and the container in this option are open source and unencumbered by patents or royalty restrictions, which makes them ideal for web distribution, but some say the quality is inferior to other options.
- **MPEG-4 container + H.264 video codec + AAC audio codec.** This combination is generally referred to as "MPEG-4," and it takes the *.mp4* or *.m4v* file suffix. H.264 is a high-quality and flexible video codec, but it is patented and must be licensed for a fee. The royalty requirement has been a deal-breaker for browsers that refuse to support it.
- **WebM container + VP8 video codec + Vorbis audio codec.** "WebM" is the newest container format and uses the *.webm* file extension. It is designed to work with VP8 and Vorbis exclusively, and has the advantage of being open source and royalty-free.

Of course, the problem that I referred to earlier is that browser makers have not agreed on a single format to support. Some go with open source, royalty-free options like Ogg Theora or WebM. Others are sticking with the better quality of H.264 despite the royalty requirements. What that means is that we web developers need to make multiple versions of videos to ensure support across all browsers. Table 10-4 lists which browsers support the various video options.

For Further Reading: HTML5 Media

I recommend these books when you are ready to learn more about HTML5 media:

- *HTML5 Media*, by Shelley Powers (O'Reilly).
- *HTML5, Up and Running*, by Mark Pilgrim (O'Reilly) includes a helpful section on HTML5 video.
- *The Definitive Guide to HTML5 Video*, by Sylvia Pfeiffer (Apress).

Table 10-4. Video support in current browsers (as of mid-2012)

Format	Type	IE	Chrome	Firefox	Safari	Opera Mobile	Mobile Safari	Android
Ogg Theora	video/ogg	–	5.0+	3.5+	–	10.5+	–	–
MP4/H.264	video/mp4	9.0+	–	–	3.1+	–	3.0+	2.0+
WebM	video/webm	9.0+	6.0+	4.0+	–	11+	–	2.3.3+

Obras protegidas por Direitos de Autor

Meet the audio formats

The landscape looks similar for audio formats: several to choose from, but no format that is supported by all browsers (Table 10-5).

- **MP3.** The MP3 format is a codec and container in one, with the file extension *.mp3*. It has become ubiquitous as a music download format. The MP3 (short for MPEG-1 Audio Layer 3) is patented and requires license fees paid by hardware and software companies (not media creators).
- **WAV.** The WAV format (*.wav*) is also a codec and container in one.
- **Ogg container + Vorbis audio codec.** This is usually referred to as "Ogg Vorbis" and is served with the *.ogg* or *.oga* file extension.
- **MPEG 4 container + AAC audio codec.** "MPEG4 audio" (*.m4a*) is less common than MP3.
- **WebM container + Vorbis audio codec.** The WebM (*.webm*) format can also contain audio only.

Table 10-5. Audio support in current browsers (as of 2012)

Format	Type	IE	Chrome	Firefox	Safari	Opera Mobile	Mobile Safari	Android
MP3	audio/mpeg	9.0+	5.0+	–	4+	–	3.0+	2.0+
WAV	audio/wav or audio/wave	–	5.0+	3.5+	4+	10.5+	3.0+	2.0+
Ogg Vorbis	audio/ogg	–	5.0+	3.5+	–	10.5+	–	2.0+
MPEG-4/AAC	audio/mp4	9.0+	5.0+	–	4+	–	3.0+	2.0+
WebM	audio/webm	9.0+	6.0+	4.0+	–	11+	–	2.3.3+

Video and Audio Encoding Tools

There are scores of options for editing and encoding video and audio files, so I can't cover them all here, but the following tools are free and get the job done.

Video conversion

- Miro Video Converter (*www.mirovideoconverter.com*) is a free tool that converts any video to H.264, Ogg Theora, or WebM format optimized for mobile devices or the desktop with a simple drag-and-drop interface. It is available for OS X and Windows.
- Handbrake (*handbrake.fr*) is a popular open source tool for getting better control over H.264 settings. It is available for Windows, OS X, and Linux.
- Firefogg (*firefogg.org*) is an extension to Firefox for converting video to the Ogg Theora format. Simply install the Firefogg extension to Firefox 3.5+, then visit the Firefogg site and convert video using its online interface.

Audio conversion

- MP3/WMA/Ogg Converter (*www.freemp3wmaconverter.com*) is a free tool that converts the following audio formats: MP3, WAV, WMA, OGG, AAC, and more. Sorry, Mac users; it is Windows only.
- On the Mac, try Max, an open source audio converter available at *sbooth.org/Max/*. Audacity (*audacity.sourceforge.net/*) also has some basic conversion tools in addition to being a recording tool.

Obras protegidas por Direitos de Autor

Adding a video to a page

`<video>...</video>`

Adds a video player to the page

NEW IN HTML5

I guess it's about time we got to the markup for adding a video to a web page (this is the HTML part of the book, after all). Let's start with an example that assumes you are designing for an environment where you know exactly what browser your user will be using. When this is the case, you can provide only one video format using the `src` attribute in the `video` tag (just as you do for an `img`). Figure 10-2 shows a movie with the default player in the Chrome browser. We'll look at the other attributes after the example.

```
<video src="highlight_reel.mp4" width="640" height="480"
poster="highlight_still.jpg" controls autoplay>
</video>
```

Figure 10-2. An embedded movie using the `video` element (shown in Chrome on Mac).

There are some juicy attributes in that example worth looking at in detail.

`width``="`*`pixel measurement`*`"`
`height``="`*`pixel measurement`*`"`
: Specifies the size of the box the embedded media player takes up on the screen. Generally, it is best to set the dimensions to exactly match the pixel dimensions of the movie. The movie will resize to match the dimensions set here.

`poster``="`*`url of image`*`"`
: Provides the location of a still image to use in place of the video before it plays.

WARNING

iOS 3 devices will not play a video that includes the `poster` attribute, so avoid using it if you need to support old iPhones and iPads.

`controls`
: Adding the **`controls`** attribute prompts the browser to display its built-in media controls, generally a play/pause button, a "seeker" that lets you move to a position within the video, and volume controls. It is possible to

Obras protegidas por Direitos de Autor

create your own custom player interface using CSS and JavaScript if you want more consistency across browsers. How to do that is beyond the scope of this chapter, but is explained in the resources listed in the "For Further Reading: HTML5 Media" sidebar. In many instances, the default controls are just fine.

`autoplay`
: Makes the video start playing automatically once it has downloaded enough of the media file to play through without stopping. In general, use of `autoplay` should be avoided in favor of letting the user decide when the video should start.

In addition, the `video` (and `audio`) element can use the `loop` attribute to make the video play again once it has finished (ad infinitum), `muted` for playing the video track without the audio, `mediagroup` for making a `video` element part of a group of related media elements (such as a `video` and a synced sign language translation), and `preload` for suggesting to the browser whether the video data should be fetched as soon as the page loads (`preload="auto"`) or wait until the user presses the play button (`preload="none"`). Setting `preload="metadata"` loads information about the media file, but not the media itself. A device can decide how to best handle the `auto` setting; for example, a browser in a smartphone may protect a user's data usage by not preloading media, even when it is set to `auto`.

Video for all!

But wait a minute! We already know that one video format isn't going to cut it in the real world. At the very least, you need to make two versions of your video: Ogg Theora and MPEG-4 (H.264 video). Some developers prefer WebM instead of Ogg because browser support is nearly as good and the files are smaller. As a fallback for users with browsers that don't support HTML5 video, you can embed a Flash player on the page or use a service like YouTube or Vimeo, in which case you let them handle the conversion, and you just copy the embed code.

In the markup, a series of `source` elements inside the `video` element point to each video file. Browsers look down the list until they find one they support and download only that version. The Flash fallback gets added with the traditional `object` and `embed` elements, so if a browser can't make heads or tails of `video` and `source`, chances are high it can play it in Flash. Finally, to ensure accessibility for all, it is highly recommended that you add some simple links to download the videos so they can be played in whatever media player is available, should all of the above fail.

Without further ado, here is one (very thorough) code example for embedding video that should serve all users, including those on mobile devices. You may choose not to provide all these formats, so adapt it accordingly.

object and embed

The `object` element is the generic way to embed media such as a movie, Flash movie, applet, even images in a web page. It contains a number of `param` (for parameters) elements that provide instructions or resources that the object needs to display. You can also put fallback content inside the `object` element that is used if the media is not supported. The attributes and parameters vary by object type and are sometimes specific to the third-party plug-in displaying the media.

The `object`'s poor cousin, `embed`, also embeds media on web pages. It has been a non-standard, but widely supported, element until it was finally made official in HTML5. Some media require the use of `embed`, which is often used as a fallback in an `object` element to appease all browsers.

You can see an example of the `object` and `param` elements in the "Video for Everybody" code example on the following page.

Obras protegidas por Direitos de Autor

The following example is based on the code in Kroc Camen's article "Video for Everybody" (*camendesign.com/code/video_for_everybody*). I highly recommend checking that page for updates, instructions for modifying the code, and many more technical details. We'll look at each part following the example.

```
<video id="yourmovieid" width="640" height="360" poster="yourmovie_
still.jpg" controls preload="auto">
    <source src="yourmovie-baseline.mp4" type='video/mp4;
codecs="avc1.42E01E, mp4a.40.2"'>
    <source src="yourmovie.webm" type='video/webm; codecs="VP8,
vorbis"'>
    <source src="yourmovie.ogv" type='video/ogg; codecs="theora,
vorbis"'>
<!--Flash fallback -->
    <object width="640" height="360" type="application/x-shockwave-
flash" data="your_flash_player.swf">
        <param name="movie" value="your_flash_player.swf">
        <param name="flashvars" value="controlbar=over&amp;image=poster.
jpg&amp;file=yourmovie-main.mp4">
        <img src="poster.jpg" width="640" height="360" alt=""
        title="No video playback capabilities, please download the video
below">
    </object>
</video>
<p>Download the Highlights Reel:</p>
<ul>
    <li><a href="yourmovie.mp4">MPEG-4 format</a></li>
    <li><a href="yourmovie.ogv">Ogg Theora format</a></li>
</ul>
```

Each **source** element contains the location of the media file (**src**) and information about its file type (**type**). In addition to listing the MIME type of the file container (e.g., **video/ogg**), it is helpful to also list the codecs that were used (see the note). This is especially important for MPEG-4 video because the H.264 codec has a number of different profiles, such as baseline (used by mobile devices), main (used by desktop Safari and IE9+), extended, and high (these two are generally not used for web video). Each profile has its own profile ID, as you see in the first **source** element in the example.

Technically, the order of the **source** elements doesn't matter, but to compensate for a bug on early iPads, it is best to put the baseline MPEG-4 first in the list. iPads running iOS 3 won't find it if it's further down, and it won't hurt any other browsers.

After the **source** elements, an **object** element is used to embed a Flash player that will play the MPEG-4 video for browsers that have the Flash plug-in. There are many Flash players available, but Kroc Camen (of "Video for Everybody" fame) recommends JW Player, which is easy to install (just put a JavaScript *.js* file and the Flash *.swf* file on your server). Download the JW Player and instructions for installing and configuring it at *www.longtailvideo.com/players/jw-flv-player/*. If you use the JW Player, replace *your_flash_player.swf* in the example with **player.swf**.

NOTE

If you look carefully, you'll see that single quotation marks (') were used to enclose the long string of values for the **type** *attribute in the* **source** *element. That is because the codecs must be enclosed in double quotation marks, so the whole attribute requires a different quotation mark type.*

NOTE

In this example, the MPEG-4 video is provided at "baseline" quality in order to play on iOS 3 devices. If iOS 3 is obsolete when you are reading this or does not appear in your traffic data, you can provide the higher-quality "main" profile version instead:

```
<source src="yourmovie-
main.mp4" type='video/mp4;
codecs="avc1.4D401E, mp4a.40.2"'>
```


Obras protegidas por Direitos de Autor

It is important to note that the Flash fallback is for browsers that do not recognize the **video** element. If a browser does support **video** but simply does not support one of the media file formats, it will *not* display the Flash version. It shows nothing. That's why it is a good idea to have direct links (**a**) to the video options outside the **video** element for maximum accessibility.

WARNING

If your server is not configured to properly report the video type (its MIME type) of your video and audio files, some browsers will not play them. The MIME types for each format are listed in the "Type" column in Tables 10-4 and 10-5. So be sure to notify your server administrator or hosting company's technical help if you intend to serve media files and get the MIME types set up correctly.

Finally, if you want the video to start playing automatically, add the **autoplay** attribute to the **video** element and **autostart=true** to the Flash **param** element like this:

```
<video src="movie.mp4" width="640" height="480" autoplay>

<param name="flashvars" value="autostart=true&amp;controlbar=over&amp;
image=poster.jpg&amp; file=yourmovie-main.mp4">
```

Keep in mind that videos will not play automatically on iOS devices, even if you set it in the code. Apple disables **autoplay** on its mobile devices to prevent unintended data transfer.

Adding audio to a page

`<audio>...</audio>`

Adds an audio file to the page

NEW IN HTML5

If you've wrapped your head around the **video** markup example, you already know how to add audio to a page. The **audio** element uses the same attributes as the **video** element, with the exception of **width**, **height**, and **poster** (because there is nothing to display). Just like the **video** element, you can provide a stack of audio format options using the **source** element, as shown in the example here.

```
<audio id="soundtrack" controls preload="auto">
  <source src="soundtrack.mp3" type="audio/mp3">
  <source src="soundtrack.ogg" type="audio/ogg">
  <source src="soundtrack.webm" type="audio/webm">
</audio>
<p>Download the Soundtrack song:</p>
<ul>
  <li><a href="soundtrack.mp3">MP3</a></li>
  <li><a href="soundtrack.ogg">Ogg Vorbis</a></li>
</ul>
```

If you want to be evil, you could embed audio in a page, set it to play automatically and then loop, and not provide any controls to stop it like this:

```
<audio src="soundtrack.mp3" autoplay loop></audio>
```

WARNING

Firefox versions 7 and earlier do not support the **loop** *attribute.*

But you would never, *ever* do something like that, right? *Right?!* Of course you wouldn't.

Canvas

Another cool, "Look Ma, no plug-ins!" addition in HTML5 is the **canvas** element and the associated Canvas API. The **canvas** element creates an area on a web page that you can draw on using a set of JavaScript functions for creating lines, shapes, fills, text, animations, and so on. You could use it to display an

Obras protegidas por Direitos de Autor

illustration, but what gives the **canvas** element so much potential (and has all the web development world so delighted) is that it's all generated with scripting. That means it is dynamic and can draw things on the fly and respond to user input. This makes it a nifty platform for creating animations, games, and even whole applications...all using the native browser behavior and without proprietary plug-ins like Flash.

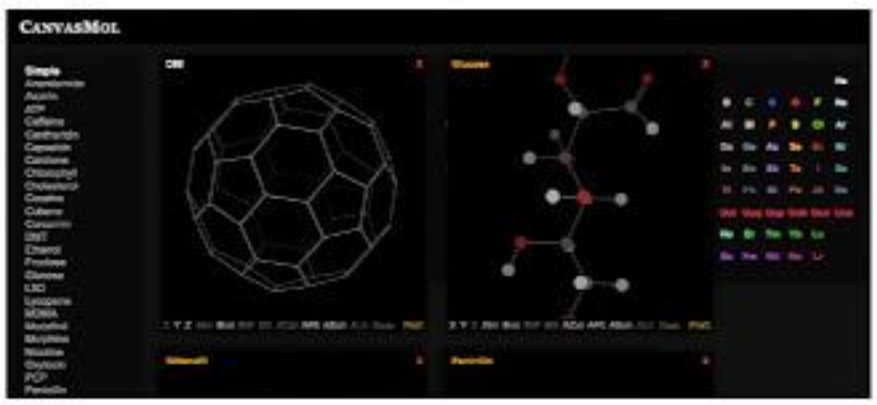
The good news is that Canvas is supported by every current browser as of this writing, with the exception of Internet Explorer 8 and earlier. Fortunately, the FlashCanvas JavaScript library (*flashcanvas.net*) can add Canvas support to those browsers using the Flash drawing API. So Canvas is definitely ready for prime time.

Figure 10-3 shows a few examples of the **canvas** element used to create games, drawing programs, an interactive molecule structure tool, and an asteroid animation. You can find more examples at *Canvasdemos.com*.

ie.microsoft.com/testdrive/Performance/AsteroidBelt/#

www.relfind.com/game/magician.html

muro.deviantart.com



alteredqualia.com/canvasmol/

Figure 10-3. A few examples of the canvas *element used for games, animations, and applications.*

Mastering the **canvas** element is more than we can take on here, particularly without any JavaScript experience under our belts, but I will give you a taste of what it is like to draw with JavaScript. That should give you a good idea of how it works, and also a new appreciation for the complexity of some of those examples.

Obras protegidas por Direitos de Autor

The canvas element

`<canvas>...</canvas>`

Adds a 2-D dynamic drawing area

NEW IN HTML5

You add a canvas space to the page with the **canvas** element and specify the dimensions with the **width** and **height** attributes. And that's really all there is to the markup. For browsers that don't support the **canvas** element, you can provide some fallback content (a message, image, or whatever seems appropriate) inside the tags.

```
<canvas width="600" height="400" id="my_first_canvas">
  Your browser does not support HTML5 canvas. Try using Chrome,
Firefox, Safari or Internet Explorer 9.
</canvas>
```

The markup just clears a space on which the drawing will happen.

Drawing with JavaScript

The Canvas API includes functions for creating basic shapes (such as **strokeRect()** for drawing a rectangular outline and **beginPath()** for starting a line drawing) and moving things around (such as **rotate()** and **scale()**), plus attributes for applying styles (for example, **lineWidth**, **strokeStyle**, **fillStyle**, and **font**).

The following example was created by my O'Reilly Media colleague Sanders Kleinfeld for his book *HTML5 for Publishers* (O'Reilly). He was kind enough to allow me to use it in this book.

Figure 10-4. The finished product of our "Hello Canvas" canvas example. See the original at examples.oreilly.com/0636920022473/my_first_canvas/my_first_canvas.html.

Figure 10-4 shows the simple smiley face we'll be creating with the Canvas API.

And here is the script that created it. Don't worry that you don't know any JavaScript yet. Just skim through the script and pay attention to the little notes. I'll also describe some of the functions in use at the end. I bet you'll get the gist of it just fine.

```
<script type="text/javascript">
window.addEventListener('load', eventWindowLoaded, false);
function eventWindowLoaded() {
    canvasApp();
}

function canvasApp(){
var theCanvas = document.getElementById('my_first_canvas');
var my_canvas = theCanvas.getContext('2d');
my_canvas.strokeRect(0,0,200,225)
    // to start, draw a border around the canvas

    //draw face
my_canvas.beginPath();
my_canvas.arc(100, 100, 75, (Math.PI/180)*0, (Math.PI/180)*360, false);
    // circle dimensions
my_canvas.strokeStyle = "black"; // circle outline is black
my_canvas.lineWidth = 3; // outline is three pixels wide
my_canvas.fillStyle = "yellow"; // fill circle with yellow
my_canvas.stroke(); // draw circle
my_canvas.fill(); // fill in circle
```


Obras protegidas por Direitos de Autor

```
my_canvas.closePath();

    // now, draw left eye
my_canvas.fillStyle = "black"; // switch to black for the fill
my_canvas.beginPath();
my_canvas.arc(65, 70, 10, (Math.PI/180)*0, (Math.PI/180)*360, false);
    // circle dimensions
my_canvas.stroke(); // draw circle
my_canvas.fill(); // fill in circle
my_canvas.closePath();

    // now, draw right eye
my_canvas.beginPath();
my_canvas.arc(135, 70, 10, (Math.PI/180)*0, (Math.PI/180)*360, false);
    // circle dimensions
my_canvas.stroke(); // draw circle
my_canvas.fill(); // fill in circle
my_canvas.closePath();

    // draw smile
my_canvas.lineWidth = 6; // switch to six pixels wide for outline
my_canvas.beginPath();
my_canvas.arc(99, 120, 35, (Math.PI/180)*0, (Math.PI/180)*-180, false);
    // semicircle dimensions
my_canvas.stroke();
my_canvas.closePath();

    // Smiley Speaks!
my_canvas.fillStyle = "black"; // switch to black for text fill
my_canvas.font      = '20px _sans'; // use 20 pixel sans serif font
my_canvas.fillText  ("Hello Canvas!", 45, 200); // write text
}
</script>
```

Finally, here is a little more information on the Canvas API functions used in the example:

strokeRect(*x1, y1, x2, y2*)
: Draws a rectangular outline from the point (x1, y1) to (x2, y2). By default, the origin of the Canvas (0,0) is the top-left corner, and *x* and *y* coordinates are measured to the right and down.

beginPath()
: Starts a line drawing.

closePath()
: Ends a line drawing that was started with **beginPath()**.

arc(*x, y, arc_radius, angle_radians_beg, angle_radians_end*)
: Draws an arc where (x,y) is the center of the circle, **arc_radius** is the length of the radius of the circle, and **angle_radians_beg** and **_end** indicate the beginning and end of the arc angle.

stroke()
: Draws the line defined by the path. If you don't include this, the path won't appear on the canvas.

Obras protegidas por Direitos de Autor

`fill()`
: Fills in the path specified with `beginPath()` and `endPath()`.

`fillText(your_text, x1, y1)`
: Adds text to the canvas starting at the (x,y) coordinate specified.

In addition, the following attributes were used to specify colors and styles:

`lineWidth`
: Width of the border of the path.

`strokeStyle`
: Color of the border.

`fillStyle`
: Color of the fill (interior) of the shape created with the path.

`font`
: The font and size of the text.

Of course, the Canvas API includes many more functions and attributes than we've used here. For a complete list, see the W3C's HTML5 Canvas 2D Context specification at *dev.w3.org/html5/2dcontext/*. A web search will turn up lots of Canvas tutorials should you be ready to learn more. In addition, I can recommend these resources:

- The book *HTML5 Canvas* by Steve Fulton and Jeff Fulton (O'Reilly)
- If watching a video is more your speed, try this tutorial: *Client-side Graphics with HTML5 Canvases: An O'Reilly Breakdown* (*shop.oreilly.com/product/0636920016502.do*)

Final Word

By now you should have a good idea of what's up with HTML5. We've looked at new elements for adding improved semantics to documents. You got a whirlwind tour of the various APIs in development that will move some useful functionality into the native browser behavior. You learned how to use the `video` and `audio` elements to embed media on the page (plus a primer on media formats). And finally, you got a peek at the `canvas` element.

In the next part of this book, CSS for Presentation, you'll learn how to write style sheets that customize the look of the page, including text styles, colors, backgrounds, and even page layout. Goodbye, default browser styles!

Obras protegidas por Direitos de Autor

Test Yourself

Let's see if you were paying attention. These questions should test whether you got the important highlights of this chapter. Good luck! And as always, the answers are in Appendix A.

1. What is the difference between HTML and XHTML?

2. Using the "XHTML Markup Requirements" sidebar as a guide, rewrite these HTML elements in XHTML syntax.

   a. `<H1> … </H1>`

   b. `<img src=image.png>`

   c. `<input type="radio" checked>`

   d. `<hr>`

   e. `<title>Sifl & Olly</title>`

   f.
   ```
   <ul>
    <li>popcorn
    <li>butter
    <li>salt
   </ul>
   ```

3. What is a DTD?

4. Name at least three ways that HTML5 is unique as a specification.

5. What is a "global attribute"?

Obras protegidas por Direitos de Autor

6. Match the API with its function:

Web Workers _____	a. Makes longitude and latitude information available
Editing API _____	b. Holds a line of communication open between the server and browser
Geolocation API _____	c. Makes web apps work even when there is no Internet connection
Web Socket _____	d. Runs processor-intensive scripts in the background
Offline Applications _____	e. Provides a set of commands for copying, pasting, and text formatting

7. Identify each of the following as a container format, video codec, or audio codec.

Ogg	_____
H.264	_____
VP8	_____
Vorbis	_____
WebM	_____
Theora	_____
AAC	_____
MPEG-4	_____

8. List the two Canvas API functions for drawing a rectangle and filling it with red. You don't need to write the whole script.

Obras protegidas por Direitos de Autor

CSS FOR PRESENTATION

PART III

IN THIS PART

Chapter 11
Cascading Style Sheets Orientation

Chapter 12
Formatting Text (Plus More Selectors)

Chapter 13
Colors and Backgrounds (Plus Even More Selectors and External Style Sheets)

Chapter 14
Thinking Inside the Box (Padding, Borders, and Margins)

Chapter 15
Floating and Positioning

Chapter 16
Page Layout with CSS

Chapter 17
Transitions, Transforms, and Animation

Chapter 18
CSS Techniques

Obras protegidas por Direitos de Autor

Obras protegidas por Direitos de Autor

image not available

Come to think of it, there really aren't any disadvantages to using style sheets. There are some lingering hassles from browser inconsistencies, but they can either be avoided or worked around if you know where to look for them.

The power of CSS

We're not talking about minor visual tweaks here, like changing the color of headlines or adding text indents. When used to its full potential, CSS is a robust and powerful design tool. My eyes were first opened to the possibilities of using CSS for design by the variety and richness of the designs at CSS Zen Garden (*www.csszengarden.com*).

In the misty days of yore, when developers were still hesitant to give up their table-based layouts for CSS, David Shea's CSS Zen Garden site demonstrated exactly what could be accomplished using CSS alone. David posted an HTML document and invited designers to contribute their own style sheets that gave the document a visual design. Figure 11-1 shows just a few of my favorites. All of these designs use the *exact same* HTML source document.

CSS Zen Dragen
by Matthew Buchanan

Shaolin Yokobue
by Javier Cabrera

By the Pier
by Peter OngKelmscott

Organica Creativa
by Eduardo Cesario

Figure 11-1. These pages from the CSS Zen Garden use the same XHTML source document, but the design is changed using exclusively CSS (used with permission of CSS Zen Garden and the individual designers).

Obras protegidas por Direitos de Autor

Not only that, they don't include a single `img` element (all of the images are used as backgrounds). But look at how different each page looks—and how sophisticated. That's all done with style sheets. It is proof of the power in keeping CSS separate from HTML, and presentation separate from structure.

The CSS Zen Garden is no longer being updated and now is considered a historical document of a turning point in the adoption of web standards. Despite its age, I still find it to be a nice one-stop lesson for demonstrating exactly what CSS can do.

Granted, it takes a lot of practice to be able to create CSS layouts like those shown in Figure 11-1. Killer graphic design skills help too (unfortunately, you won't get those in this book). I'm showing this to you up front because I want you to be aware of the potential of CSS-based design, particularly because the examples in this beginners' book tend to be simple and straightforward. Take your time learning, but keep your eye on the prize.

How Style Sheets Work

It's as easy as 1-2-3!

1. Start with a document that has been marked up in HTML.
2. Write style rules for how you'd like certain elements to look.
3. Attach the style rules to the document. When the browser displays the document, it follows your rules for rendering elements (unless the user has applied some mandatory styles, but we'll get to that later).

OK, so there's a bit more to it than that, of course. Let's give each of these steps a little more consideration.

1. Marking up the document

You know a lot about marking up content from the previous chapters. For example, you know that it is important to choose elements that accurately describe the meaning of the content. You've also heard me say that the markup creates the structure of the document, sometimes called the structural layer, upon which the presentation layer can be applied.

In this and the upcoming chapters, you'll see that having an understanding of your document's structure and the relationships between elements is central to your work as a style sheet author.

To get a feel for how simple it is to change the look of a document with style sheets, try your hand at Exercise 11-1. The good news is that I've whipped up a little HTML document for you to play with.

Obras protegidas por Direitos de Autor

image not available

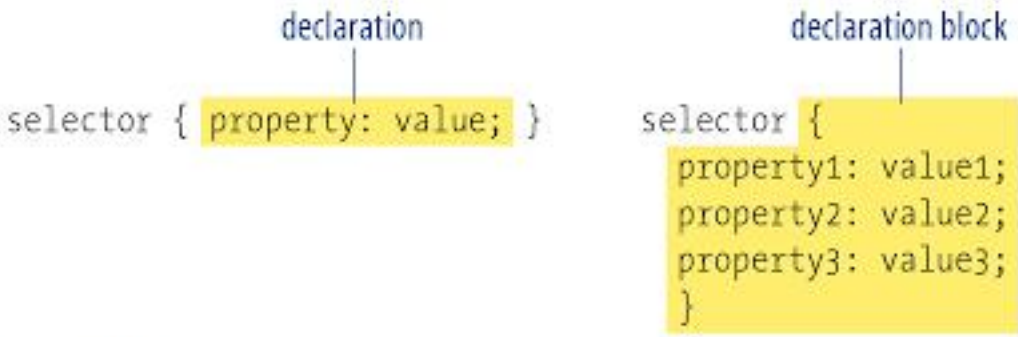


Figure 11-3. The parts of a style sheet rule.

Selectors

In the previous small style sheet example, the **h1** and **p** elements are used as selectors. This is called an element type selector, and it is the most basic type of selector. The properties defined for each rule will apply to every **h1** and **p** element in the document, respectively. In upcoming chapters, I'll introduce you to more sophisticated selectors that you can use to target elements, including ways to select groups of elements and elements that appear in a particular context.

Mastering selectors—that is, choosing the best type of selector and using it strategically—is an important step in becoming a CSS Jedi Master.

Declarations

The declaration is made up of a property/value pair. There can be more than one declaration in a single rule; for example, the rule for the **p** element shown earlier in the code example has both the **font-size** and **font-family** properties. Each declaration must end with a semicolon to keep it separate from the following declaration (see note). If you omit the semicolon, the declaration and the one following it will be ignored. The curly brackets and the declarations they contain are often referred to as the declaration block (Figure 11-3).

Because CSS ignores whitespace and line returns within the declaration block, authors typically write each declaration in the block on its own line, as shown in the following example. This makes it easier to find the properties applied to the selector and to tell when the style rule ends.

```
p {
  font-size: small;
  font-family: sans-serif;
}
```

Note that nothing has really changed here—there is still one set of curly brackets, semicolons after each declaration, etc. The only difference is the insertion of line returns and some character spaces for alignment.

The heart of style sheets lies in the collection of standard properties that can be applied to selected elements. The complete CSS specification defines dozens of properties for everything from text indents to how table headers should be read aloud. This book covers the most common and best-supported properties that you can begin using right away.

NOTE

Technically, the semicolon is not required after the last declaration in the block, but it is recommended that you get into the habit of always ending declarations with a semicolon. It will make adding declarations to the rule later that much easier.

Providing Measurement Values

When you're providing measurement values, the unit must immediately follow the number, like this:

```
margin: 2em;
```

Adding a space before the unit will cause the property not to work.

INCORRECT: `margin: 2 em;`

It is acceptable to omit the unit of measurement for zero values:

```
margin: 0;
```


Obras protegidas por Direitos de Autor

Values are dependent on the property. Some properties take length measurements, some take color values, and others have a predefined list of keywords. When using a property, it is important to know which values it accepts; however, in many cases, simple common sense will serve you well.

Before we move on, why not get a little practice writing style rules yourself in the continuation of Exercise 11-1?

exercise 11-1 | Your first style sheet (continued)

Open *twenties.html* in a text editor. In the **head** of the document you will find that I have set up a **style** element for you to type the rules into. The **style** element is used to embed a style sheet in an HTML document.

To begin, we'll simply add the small style sheet that we just looked at in this section. Type the following rules into the document, just as you see them here:

```
<style type="text/css">
h1 {
  color: green;
}
p {
  font-size: small;
  font-family: sans-serif;
}
</style>
```

Save the file, and take a look at it in the browser. You should notice some changes (if your browser already uses a sans-serif font, you may only see a size change). If not, go back and check that you included both the opening and closing curly bracket and semicolons. It's easy to accidentally omit these characters, causing the style sheet not to work.

Now we'll change and add to the style sheet to see how easy it is to write rules and see the effects of the changes. Here are a few things to try (remember that you need to save the document after each change in order for the changes to be visible when you reload it in the browser).

- Make the **h1** element "gray" and take a look at it in the browser. Then make it "blue". Finally, make it "red". (We'll run through the complete list of available color names in Chapter 13, Colors and Backgrounds.)
- Add a new rule that makes the **h2** elements red as well.
- Add a 100-pixel left margin to paragraph (**p**) elements using this declaration:

  ```
  margin-left: 100px;
  ```

Remember that you can add this new declaration to the existing rule for **p** elements.

- Add a 100-pixel left margin to the **h2** headings as well.
- Add a red, 1-pixel border to the bottom of the **h1** element using this declaration:

  ```
  border-bottom: 1px solid red;
  ```

- Move the image to the right margin, and allow text to flow around it with the **float** property. The shorthand **margin** property shown in this rule adds zero pixels of space on the top and bottom of the image and 12 pixels of space on the left and right of the image (the values are mirrored in a manner explained in Chapter 14, Thinking Inside the Box).

  ```
  img {
    float: right;
    margin: 0 12px;
  }
  ```

When you are done, the document should look something like the one shown in Figure 11-4.

Figure 11-4. The article after adding the small style sheet from the example. As I said, not beautiful—just different.

Obras protegidas por Direitos de Autor

3. Attaching the styles to the document

In the previous exercise, we embedded the style sheet right in the document using the **style** element. That is just one of three ways that style information can be applied to an HTML document. You'll get to try out each of these soon, but it is helpful to have an overview of the methods and terminology up front.

External style sheets. An external style sheet is a separate, text-only document that contains a number of style rules. It must be named with the *.css* suffix. The *.css* document is then linked to or imported into one or more HTML documents (we'll discuss how in Chapter 13). In this way, all the files in a website may share the same style sheet. This is the most powerful and preferred method for attaching style sheets to content.

Embedded style sheets. This is the type of style sheet we worked with in the exercise. It is placed in a document using the **style** element, and its rules apply only to that document. The **style** element must be placed in the **head** of the document. This example also includes a comment (see the "Comments in Style Sheets" sidebar).

```
<head>
   <title>Required document title here</title>
     <style>
     /* style rules go here */
    </style>
</head>
```

Comments in Style Sheets

Sometimes it is helpful to leave yourself or your collaborators comments in a style sheet. CSS has its own comment syntax, shown here:

```
/* comment goes here */
```

Content between the **/*** and ***/** will be ignored when the style sheet is parsed, which means you can leave comments anywhere in a style sheet, even within a rule.

```
body {
  font-size: small;
 /* font-size:large; */
}
```

One use for comments is to label sections of the style sheet to make things easier to find later; for example:

```
/* Layout styles */
```

or:

```
/* FOOTER STYLES */
```

CSS comments are also useful for temporarily hiding style declarations in the design process. When I am trying out a number of styles, I can quickly switch styles off by enclosing them in **/*** and ***/**, check it in a browser, then remove the comment characters to make the style appear again. It's much faster than retyping the entire thing.

NOTE

In HTML 4.01 and XHTML 1.0/1.1, the **style** *element must contain a* **type** *attribute that identifies the content of the* **style** *element:*

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

In HTML5, the **type** *attribute is no longer required.*

The **style** *element may also include the* **media** *attribute used to target specific media such as screen, print, or handheld devices. These are discussed in Chapter 13 as well.*

Obras protegidas por Direitos de Autor

Inline styles. You can apply properties and values to a single element using the **style** attribute in the element itself, as shown here:

```
<h1 style="color: red">Introduction</h1>
```

To add multiple properties, just separate them with semicolons, like this:

```
<h1 style="color: red; margin-top: 2em">Introduction</h1>
```

Inline styles apply only to the particular element in which they appear. Inline styles should be avoided, unless it is absolutely necessary to override styles from an embedded or external style sheet. Inline styles are problematic in that they intersperse presentation information into the structural markup. They also make it more difficult to make changes because every **style** attribute must be hunted down in the source.

Exercise 11-2 gives you an opportunity to write an inline style and see how it works. We won't be working with inline styles after this point for the reasons listed earlier, so here's your chance.

exercise 11-2 | Applying an inline style

Open the article *twenties.html* in whatever state you last left it in Exercise 11-1. If you worked to the end of the exercise, you will have a rule that makes the **h2** elements red.

Write an inline style that makes the second **h2** gray. We'll do that right in the opening **h2** tag using the **style** attribute, as shown here:

```
<h2 style="color: gray">
Connect-the-Dots</h2>
```

Save the file and open it in a browser. Now the second heading is gray, overriding the red color set in the embedded style sheet. The other **h2** heading is unaffected.

The Big Concepts

There are a few big ideas that you need to get your head around to be comfortable with how Cascading Style Sheets behave. I'm going to introduce you to these concepts now so we don't have to slow down for a lecture once we're rolling through the style properties. Each of these ideas will certainly be revisited and illustrated in more detail in the upcoming chapters.

Inheritance

Are your eyes the same color as your parents'? Did you inherit their hair color? Your unique smile? Well, just as parents pass down traits to their children, styled HTML elements pass down certain style properties to the elements they contain. Notice in Exercise 11-1, when we styled the **p** elements in a small, sans-serif font, the **em** element in the second paragraph became small and sans-serif as well, even though we didn't write a rule for it specifically (Figure 11-5). That is because it inherited the styles from the paragraph it is in.

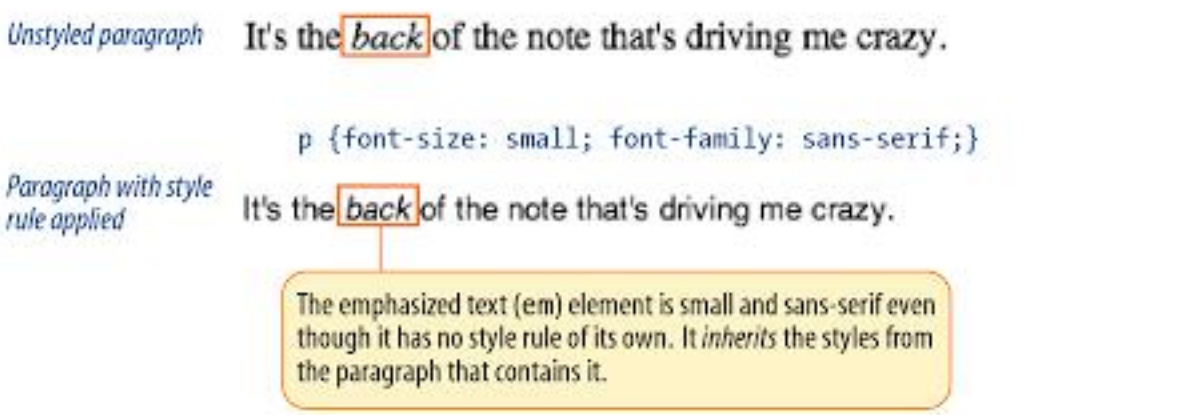


Figure 11-5. The **em** *element inherits styles that were applied to the paragraph.*

Obras protegidas por Direitos de Autor

Document structure

This is where an understanding of your document's structure becomes important. As I've noted before, HTML documents have an implicit structure or hierarchy. For example, the sample article we've been playing with has an `html` root element that contains a **head** and a **body**, and the **body** contains heading and paragraph elements. A few of the paragraphs, in turn, contain inline elements such as images (`img`) and emphasized text (`em`). You can visualize the structure as an upside-down tree, branching out from the root, as shown in Figure 11-6.

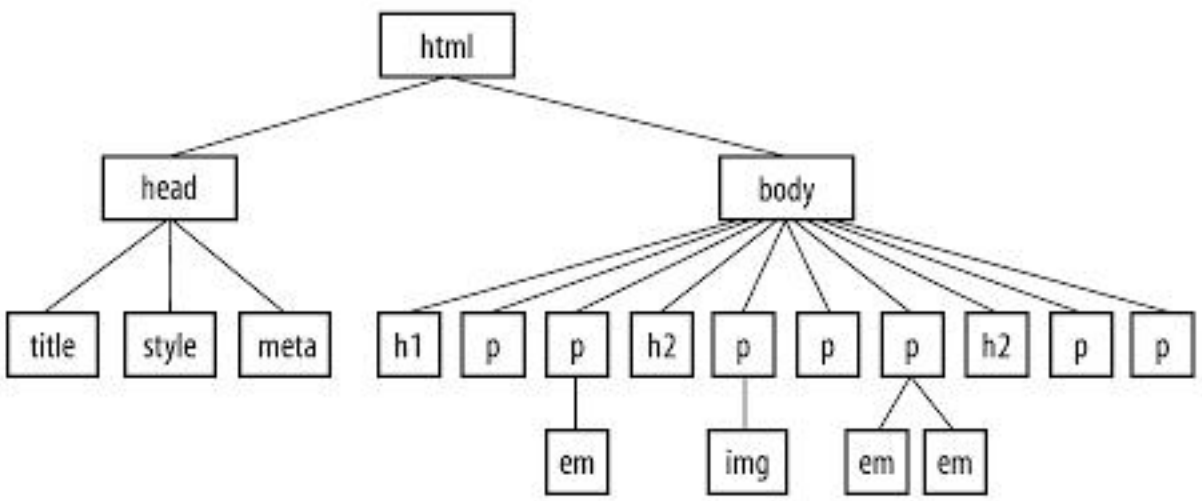


Figure 11-6. The document tree structure of the sample document, twenties.html.

Parents and children

The document tree becomes a family tree when it comes to referring to the relationship between elements. All the elements contained within a given element are said to be its descendants. For example, the `h1`, `h2`, `p`, `em`, and `img` elements in the document in Figure 11-6 are all descendants of the **body** element.

An element that is directly contained within another element (with no intervening hierarchical levels) is said to be the child of that element. Conversely, the containing element is the parent. For example, the `em` element is the child of the `p` element, and the `p` element is its parent.

All of the elements higher than a particular element in the hierarchy are its ancestors. Two elements with the same parent are siblings. We don't refer to "aunts" or "cousins," so the analogy stops there. This may all seem academic, but it will come in handy when writing CSS selectors.

Obras protegidas por Direitos de Autor

Pass it on

When you write a font-related style rule using the `p` element as a selector, the rule applies to all of the paragraphs in the document as well as the inline text elements they contain. We've seen the evidence of the `em` element inheriting the style properties applied to its parent (`p`) back in Figure 11-5. Figure 11-7 demonstrates what's happening in terms of the document structure diagram. Note that the `img` element is excluded because font-related properties do not apply to images.

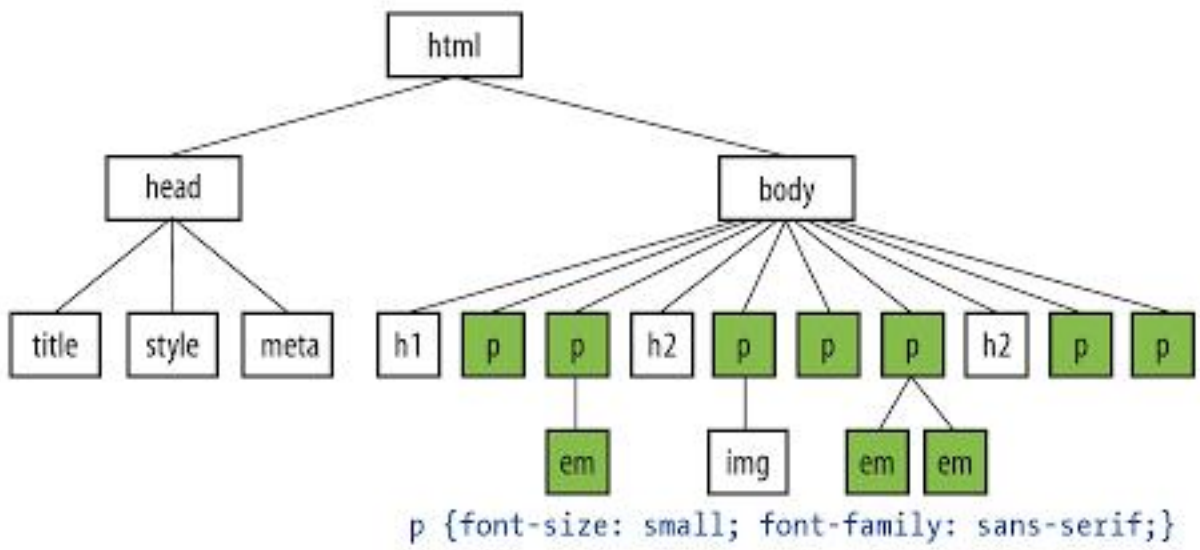


Figure 11-7. Certain properties applied to the `p` element are inherited by their children.

CSS TIP

When you learn a new property, it is a good idea to note whether it inherits. Inheritance is noted for every property listing in this book. For the most part, inheritance follows your expectations.

Notice that I've been saying "certain" properties are inherited. It's important to note that some style sheet properties inherit and others do not. In general, properties related to the styling of text—font size, color, style, etc.—are passed down. Properties such as borders, margins, backgrounds, and so on that affect the boxed area around the element tend not to be passed down. This makes sense when you think about it. For example, if you put a border around a paragraph, you wouldn't want a border around every inline element (such as `em`, `strong`, or `a`) it contains as well.

You can use inheritance to your advantage when writing style sheets. For example, if you want all text elements to be rendered in the Verdana font face, you could write separate style rules for every element in the document and set the `font-face` to Verdana. A *better* way would be to write a single style rule that applies the `font-face` property to the `body` element, and let all the text elements contained in the `body` inherit that style (Figure 11-8).

Any property applied to a specific element will override the inherited values for that property. Going back to the article example, we could specify that the `em` element should appear in a serif font, and that would override the inherited sans-serif setting.

WARNING

The browser's style sheet may override styles set on the `body`, so be on the lookout for unexpected styling.

Obras protegidas por Direitos de Autor

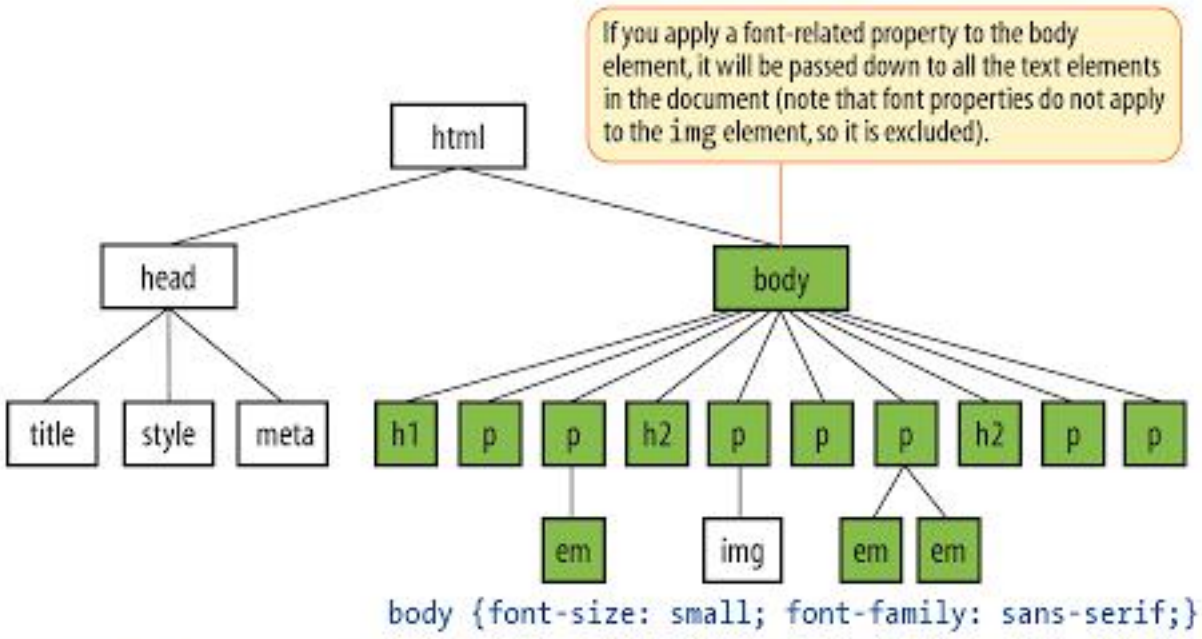


Figure 11-8. All the elements in the document inherit certain properties applied to the **body** *element.*

Conflicting styles: the cascade

Ever wonder why they are called "cascading" style sheets? CSS allows you to apply several style sheets to the same document, which means there are bound to be conflicts. For example, what should the browser do if a document's imported style sheet says that `h1` elements should be red, but its embedded style sheet has a rule that makes `h1`s purple?

The folks who wrote the style sheet specification anticipated this problem and devised a hierarchical system that assigns different weights to the various sources of style information. The cascade refers to what happens when several sources of style information vie for control of the elements on a page: style information is passed down ("cascades" down) until it is overridden by a style command with more weight.

For example, if you don't apply any style information to a web page, it will be rendered according to the browser's internal style sheet (we've been calling this the default rendering; the W3C calls it the user agent style sheet). Individual users can apply their own styles as well (the user style sheet), which overrides the default styles in their browser. However, if the author of the web page has attached a style sheet (the author style sheet), that overrides both the user and the user agent styles. The only exception is if the user has identified a style as "important," in which case that style will trump all (see the "Assigning Importance" sidebar).

Obras protegidas por Direitos de Autor

The style sheet source is one hierarchy that determines which style wins. As we've learned, there are three ways to attach style information to the source document, and they have a cascading order as well. Generally speaking, the closer the style sheet is to the content, the more weight it is given. Embedded style sheets that appear right in the document in the `style` element have more weight than external style sheets. In the example that started this section, the `h1` elements would end up purple as specified in the embedded style sheet, not red as specified in the external *.css* file that has less weight. Inline styles have more weight than embedded style sheets because you can't get any closer to the content than a style right in the element's opening tag. That's the effect we witnessed in Exercise 11-2.

To prevent a specific rule from being overridden, you can assign it "importance" with the `!important` indicator, as explained in the "Assigning Importance" sidebar. The sidebar "Style Sheet Hierarchy" provides an overview of the cascading order from general to specific.

Specificity

When two rules in a single style sheet conflict, the type of selector is used to determine the winner.

Once the applicable style sheet has been chosen, there may still be conflicts; therefore, the cascade continues at the rule level. When two rules in a single style sheet conflict, the type of selector is used to determine the winner. The more specific the selector, the more weight it is given to override conflicting declarations.

It's a little soon to be discussing specificity because we've only looked at one type of selector (and the least specific type, at that). For now, put the term specificity and the concept of some selectors overriding others on your radar. We will revisit it in Chapter 12 when you have more selector types under your belt.

Assigning Importance

If you want a rule not to be overridden by a subsequent conflicting rule, include the `!important` indicator just after the property value and before the semicolon for that rule. For example, to make paragraph text blue always, use the following rule:

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

Even if the browser encounters an inline style later in the document (which should override a document-wide style sheet), like this one:

```
<p style="color: red">
```

that paragraph will still be blue because the rule with the `!important` indicator cannot be overridden by other styles in the author's style sheet.

The only way an `!important` rule may be overridden is by a conflicting rule in a reader (user) style sheet that has also been marked `!important`. This is to ensure that special reader requirements, such as large type for the visually impaired, are never overridden.

Based on the previous examples, if the reader's style sheet includes this rule:

```
p {color: black;}
```

the text would still be blue because all author styles (even those not marked `!important`) take precedence over the reader's styles. However, if the conflicting reader's style is marked `!important`, like this:

```
p {color: black !important;}
```

the paragraphs will be black and cannot be overridden by any author-provided style.

Obras protegidas por Direitos de Autor

Style Sheet Hierarchy

Style information can come from various sources, listed here from general to specific. Items lower in the list will override items above them:

- Browser default settings
- User style settings (set in a browser as a "reader style sheet")
- Linked external style sheet (added with the `link` element)
- Imported style sheets (added with the `@import` function)
- Embedded style sheets (added with the `style` element)
- Inline style information (added with the `style` attribute in an opening tag)
- Any style rule marked `!important` by the author
- Any style rule marked `!important` by the reader (user)

Rule order

Finally, if there are conflicts within style rules of identical weight, whichever one comes last in the list "wins." Take these three rules, for example:

```
<style>
  p { color: red; }
  p { color: blue; }
  p { color: green; }
</style>
```

In this scenario, paragraph text will be green because the last rule in the style sheet—that is, the one closest to the content in the document—overrides the earlier ones. The same thing happens when conflicting styles occur within a single declaration stack:

```
<style>
  p { color: red;
      color: blue;
      color: green; }
</style>
```

The resulting color will be green because the last declaration overrides the previous two. It is easy to accidentally override previous declarations within a rule when you get into compound properties, so this is an important behavior to keep in mind.

NOTE

This "last one listed wins" rule applies in other contexts in CSS as well. For example, external style sheets listed later in the source will be given precedence over those listed above them.

The box model

As long as we're talking about "big CSS concepts," it is only appropriate to introduce the cornerstone of the CSS visual formatting system: the box model. The easiest way to think of the box model is that browsers see every element on the page (both block and inline) as being contained in a little rectangular box. You can apply properties such as borders, margins, padding, and backgrounds to these boxes, and even reposition them on the page.

Obras protegidas por Direitos de Autor

We're going to go into a lot more detail about the box model in Chapter 14, but having a general feel for it will benefit you even as we discuss text and backgrounds in the following two chapters.

To see the elements roughly the way the browser sees them, I've written style rules that add borders around every content element in our sample article.

```
h1 { border: 1px solid blue; }
h2 { border: 1px solid blue; }
p { border: 1px solid blue; }
em { border: 1px solid blue; }
img { border: 1px solid blue; }
```

Figure 11-9 shows the results. The borders reveal the shape of each block element box. There are boxes around the inline elements (`em` and `img`) as well. Notice that the block element boxes expand to fill the available width of the browser window, which is the nature of block elements in the normal document flow. Inline boxes encompass just the characters or image they contain.

Figure 11-9. Rules around all the elements reveal their element boxes.

Grouped selectors

Hey! This is a good opportunity to show you a handy style rule shortcut. If you ever need to apply the same style property to a number of elements, you can group the selectors into one rule by separating them with commas. This one rule has the same effect as the five rules listed previously. Grouping them makes future edits more efficient and results in a smaller file size.

```
h1, h2, p, em, img { border: 1px solid blue; }
```

Now you have two selector types in your toolbox: a simple element selector and grouped selectors.

Pop Quiz

Can you guess why I didn't just add the **`border`** property to the **`body`** element and let it inherit to all the elements in the grouped selector?

Answer:

Because **`border`** is one of the properties that is not inherited.

Obras protegidas por Direitos de Autor

A Quick History of CSS

The first official version of CSS (the CSS Level 1 Recommendation, a.k.a CSS1) was officially released in 1996, and included properties for adding font, color, and spacing instructions to page elements. Unfortunately, lack of dependable browser support prevented the widespread adoption of CSS for several years.

CSS Level 2 (CSS2) was released in 1998. It most notably added properties for positioning that allowed CSS to be used for page layout. It also introduced styles for other media types (such as print, handheld, and aural) and more sophisticated methods for selecting elements for styling. CSS Level 2, Revision 1 (CSS2.1) makes some minor adjustments to CSS2 and became a full Recommendation in 2011.

CSS Level 3 (CSS3) is different from prior versions in that it has been divided into many individual modules, each addressing a feature such as animation, multiple column layouts, or borders. While some modules are being standardized, others remain experimental. In that way, browser developers can begin implementing (and we can begin using!) one feature at a time instead of waiting for an entire specification to be "ready." In fact, many developers use enhanced CSS3 features even though they aren't universally supported as long as the fallback is usable and no content is lost. They can be used as "frosting" on an otherwise stable design (or in other words, as an enhancement).

To keep up to date with the various CSS features in the works, see the W3C's CSS Current work page at *www.w3.org/Style/CSS/current-work*.

Moving Forward with CSS

This chapter covered all the fundamentals of Cascading Style Sheets, including rule syntax, ways to apply styles to a document, and the central concepts of inheritance, the cascade, and the box model. Style sheets should no longer be a mystery, and from this point on, we'll merely be building on this foundation by adding properties and selectors to your arsenal as well as expanding on the concepts introduced here.

CSS is a vast topic, well beyond the scope of this book. Bookstores and the Web are loaded with information about style sheets for all skill levels. I've compiled a list of the resources I've found the most useful during my learning process. I've also provided a list of popular tools that assist in writing style sheets.

Books

There is no shortage of good books on CSS out there, but these are the ones that taught me, and I feel good recommending them.

Cascading Style Sheets: The Definitive Guide, by Eric A. Meyer (O'Reilly)

CSS: The Missing Manual, by David Sawyer McFarland (O'Reilly)

Obras protegidas por Direitos de Autor

Handcrafted CSS: More Bulletproof Web Design, by Dan Cederholm (New Riders)

CSS Cookbook: Quick Solutions to Common CSS Problems, by Christopher Schmitt (O'Reilly)

Online resources

The sites listed here are good starting points for online exploration of style sheets.

World Wide Web Consortium (*www.w3.org/Style/CSS*). The World Wide Web Consortium oversees the development of web technologies, including CSS.

A List Apart (*www.alistapart.com/topics/code/css/*). This online magazine features some of the best thinking and writing on cutting-edge, standards-based web design. It was founded in 1998 by Jeffrey Zeldman and Brian Platz.

CSS-tricks (*css-tricks.com*). The is the blog of CSS whiz kid Chris Coyier. Chris *loves* CSS and enthusiastically shares his research and tinkering on his site.

CSS tools

Here are a couple of tools that I can personally recommend.

Web Developer extension

Web developers are raving about the Web Developer extension written by Chris Pederick. The extension adds a toolbar to the browser with tools that enable you to analyze and manipulate any page in the window. You can edit the style sheet for the page you are viewing as well as get information about the HTML and graphics. It also validates the CSS, HTML, and accessibility of the page. It is available for Chrome and Firefox/Mozilla browsers. Get it at *chrispederick.com/work/web-developer*. Note that Safari has a similar built-in inspector (go to Develop → Show Web Inspector).

Web-authoring programs

Current WYSIWYG authoring programs such as Adobe Dreamweaver and Microsoft Expression Web can be configured to write a style sheet for you automatically as you design the page. The downside is that they are not always written in the most efficient manner (for example, they tend to overuse the `class` attribute to create style rules). Still, they may give you a good head start on the style sheet that you can then edit manually.

Obras protegidas por Direitos de Autor

Test Yourself

Here are a few questions to test your knowledge of the CSS basics. Answers are provided in Appendix A.

1. Identify the various parts of this style rule:

```
blockquote { line-height: 1.5; }
```

selector: _____ value: _____

property: _____ declaration: _____

2. What color will paragraphs be when this embedded style sheet is applied to a document? Why?

```
<style type="text/css">
   p { color: purple; }
   p { color: green; }
   p { color: gray; }
</style>
```

3. Rewrite each of these CSS examples. Some of them are completely incorrect, and some could just be written more efficiently.

a.
```
p {font-family: sans-serif;}
p {font-size: 1em;}
p {line-height: 1.2em;}
```

b.
```
blockquote {
 font-size: 1em
 line-height: 150%
 color: gray }
```

c.
```
body
 {background-color: black;}
 {color: #666;}
 {margin-left: 12em;}
 {margin-right: 12em;}
```

d.
```
p {color: white;}
blockquote {color: white;}
li {color: white;}
```

e.
```
<strong style="red">Act now!</strong>
```


Obras protegidas por Direitos de Autor

4. Circle all the elements that you would expect to appear in red when the following style rule is applied to a document with the structure diagrammed in Figure 11-10. This rule uses a type of selector you haven't seen yet, but common sense should serve you well.

```
div#intro { color: red;}
```

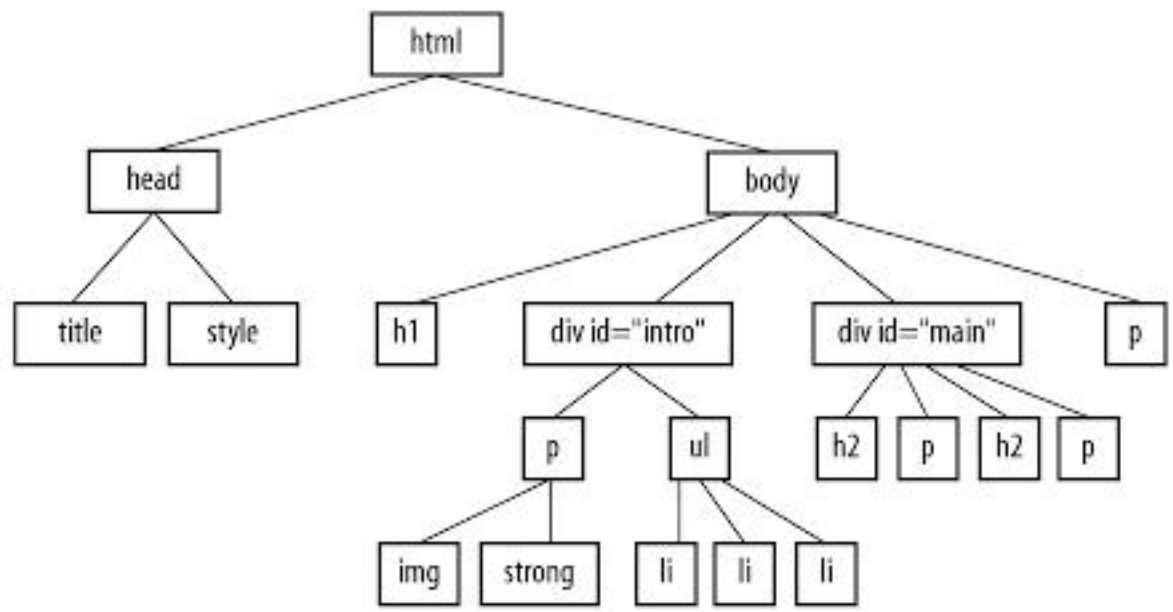


Figure 11-10. The document structure of a sample document.

Obras protegidas por Direitos de Autor

Figure 12-1. Before and after views of the Black Goose Bistro menu that we'll be working on in this chapter.

Specifying the font name

Choosing a typeface, or font family as it is called in CSS, for your text is a good place to start. Let's begin with the **font-family** property and its values.

font-family

Values: *one or more font or generic font family names, separated by commas* | inherit

Default: *depends on the browser*

Applies to: *all elements*

Inherits: *yes*

Use the **font-family** property to specify a font or list of fonts (known as a font stack) by name, as shown in these examples.

```
body { font-family: Arial; }
var { font-family: Courier, monospace; }
p { font-family: "Duru Sans", Verdana, sans-serif; }
```

Here are some important syntax requirements:

- All font names, with the exception of generic font families, must be capitalized. For example, use "Arial" instead of "arial".
- Use commas to separate multiple font names, as shown in the second and third examples.

Obras protegidas por Direitos de Autor

Say Hello to Web Fonts

The ability to provide your own font for use on a web page has been around since 1998, but it was never feasible due to browser inconsistencies. Fortunately, that story has changed, and now web fonts are a perfectly viable option. The Web has never looked better!

There is a lot to say about web fonts, so this sidebar is merely an introduction to the highlights, starting with the challenges.

What took you so long?

There have been two main hurdles to including fonts with web pages. First, there is the problem that different browsers support different font formats. Most fonts come in OpenType (OTF) or TrueType (TTF) format, but Internet Explorer only accepts its proprietary Embedded Open Type (EOT).

The good news is that there is a new standard for packaging fonts for delivery to web pages that all browser vendors, even IE, are implementing. The new format, WOFF (for Web Open Font Format), is a container that packages font files for web delivery. Now that IE9 is supporting WOFF, one day it may be all we need. As of this writing, however, we still need to provide the same font in a number of different formats (more on that in just a moment).

The other issue with providing fonts on web pages is that the font companies (also called foundries) are concerned (a nice way to say "freaked out") that their fonts will be sitting vulnerably on servers and available for download. Fonts cost a lot to create and are very valuable. Most come with licenses that cover very specific uses by a limited number of machines, and "free to download for whatever" is usually not included.

So, to link to a web font, you need to use the font legally and provide it in a way that all browsers support. There are two general approaches to providing fonts: host them yourself or use a web font service. Let's look at both options.

Host your own

In the "host your own" option, you find the font you want, put it on your server in all the required formats, and link it to your web page using the CSS3 `@font-face` rule.

Step 1: Find a font. This can be a bit of a challenge because the End User License Agreement (EULA) for virtually all commercial fonts does not cover web usage. Be sure to purchase the additional web license if it is available. However, thanks to demand, some foundries are opening fonts up for web use, and there are a growing number of open source fonts that you can use for free. The service Fontspring (*fontspring.com*), by Ethan Dunham, is a great place to purchase fonts that have a web license that you can use on your site or your own computer. The site Font Squirrel (*fontsquirrel.com*), also by Ethan Dunham, is a great source for open source fonts that can be used for commercial purposes for free.

Step 2: Save it in multiple formats. As of this writing, providing multiple formats is a reality. There are tools that will convert your source font into other formats, but there is a service that will take your font and make everything you need for you—the "@font-face Generator" from Font Squirrel (*www.fontsquirrel.com/fontface/generator*). Go to that page, upload your font, and it gives back the font in TTF, EOT, WOFF, and SVG, as well as the CSS code you need to make it work. Bear in mind that you should use the service only for a font that specifically allows web usage (whether that font is free, open source, or commercial). Note also that you will get better-quality font versions directly from a font vendor than you will using the Generator.

Step 3: Upload to the server. Developers typically keep their font files in the same directory as the CSS files, but that's just a matter of preference. If you download a package from Font Squirrel, be sure to keep the pieces together as you found them.

Step 4: Write the code. Link the font to your site using the `@font-face` rule in your *.css* document. The rule gives the font a `font-family` name that you can then reference later in your style sheet. It also lists the locations of the font files in their various formats. This cross-browser code example was developed by Ethan Dunham (yep, him again!) to address a bug in IE. I recommend reading the full article at *www.fontspring.com/blog/the-new-bulletproof-font-face-syntax*. See also Paul Irish's updated version at *paulirish.com/2009/bulletproof-font-face-implementation-syntax/*.

```
@font-face {

font-family: 'Font_name';
  src: url('myfont-webfont.eot?#iefix')
    format('embedded-opentype'),
  url('myfont-webfont.woff') format('woff'),
  url('myfont-webfont.ttf')  format('truetype'),
  url('myfont-webfont.svg#svgFontName')
    format('svg');
}
```

Then you just refer to the established font name in your font rules, like so:

```
p {font-family: Font_name; }
```

Use a font embedding service

If that seems like a lot of work, you may want to sign up with one of the font embedding services that do all the heavy lifting for you. For a fee, you get access to high-quality fonts, and the service handles font licensing and font protection for the foundries. They also generally provide an interface and tools that make embedding a font as easy as copy and paste.

Obras protegidas por Direitos de Autor

INDEX

Symbols

3-D transforms, 418–421
  functions for, 419
8-bit, 511
8-bit indexed color images, 517
24-bit Truecolor images, 515
& (ampersand)
  beginning character references, 99
  in XHTML, 99
<!-- and --> tags denoting comments, 54, 60
< > (angle brackets)
  > (greater than) operator, 468, 481
  >= (greater than or equal to) operator, 468
  in HTML tags, 26, 28, 55, 66
  < (less than) operator, 468
  <= (less than or equal to) operator, 468, 472
* (asterisk)
  multiplication operator, 469
  universal selector, 246
\ (backslash) versus / (slash), 55
: (colon), double and single colon indicating CSS pseudo-elements, 279
@counter-style rule (CSS), 259
{ } (curly braces)
  enclosing functions, 474
  enclosing statement blocks, 470, 472
. (dot)
  dot-dot-slash (../) pathname convention, 112
  indicating class selector, 246
= (equals sign)
  assignment operator, 465
  == (equal to) operator, 468
  === (identical to) operator, 468
! (exclamation mark)
  !important indicator, CSS style rules, 218
  != (not equal to) operator, 468, 470
  !== (not identical to) operator, 468
@font-face rule (CSS), 227–228
(hash mark, pound sign, or octothorpe symbol)
  before fragment name, 116
  before hex values, 283
@import rule (CSS), 302
@keyframes rule (CSS), 421
@media rule (CSS), 448
- (minus sign)
  - - (decrement) operator, 469
  subtraction operator, 469
( ) (parentheses) in functions, 474
. (period). *See* . (dot)
+ (plus sign)
  += (addition and assignment) operator, 469
  addition operator, 469
  ++ (increment) operator, 469, 472
  string concatenation operator, 467
" " (quotation marks, double)
  , 197
  around q elements, 89
  enclosing attribute values, 62
; (semicolon)
  ending character references, 99
  ending CSS declarations, 211
  ending JavaScript statements, 464
/ (slash)
  at start of pathname, 114
  division operator, 469
  in closing (or end) HTML tags, 55
    omitting, problems caused by, 66
  in directory names, 25
  /* */ in multi-line JavaScript comments, 465
  // in single-line JavaScript comments, 465
[ ] (square brackets), array operator, 468

A

AAC audio codec, 193
a (anchor) element, 105
  changes in HTML5, 106
  href attribute, 106–107
    mailto links, 119
    telephone links, 120
abbr (abbreviation) element, 89
absolute (measurement) units, 234
  pixels and, 236
absolute positioning, 357, 359–368
  advantage of floats over for layout, 381
  containing blocks, 360–362
  specifying position, 363–365
  stacking order, 367–368
absolute URLs, 106
accessibility, 41–42
  HTML documents, 37
  of images, 127
  SVG image content, 537
  tables, 137, 142–144
acronyms, 89
action attribute, form element, 149
active state, 278
adaptive layouts, 446
Adaptive Web Design (Gustafson), 38
addEventListener method, 480
address element, 84

Obras protegidas por Direitos de Autor

adjacent sibling selectors, 245
Adobe. *See also* individual application names
  Creative Suite, 231
  Dreamweaver, 4, 16
  Fireworks, 17, 268, 508
  Flash, 11
  Illustrator, 17, 508
  layout modules, CSS, 381
  Photoshop, 17, 268, 508
  Photoshop Elements, 17
:after pseudo-element, 280
Ajax, 497, 498
  defined, 497
A List Apart, 222
  "Elastic Designs" article, 378
  "Fluid Grids" article, 376
alpha, 271
alpha or alpha channel transparency, 518, 526
  multiple levels of, using PNG-24, 553
  PNG-8 alpha transparency, 531
  working with, 527
alt attribute, img (image) element, 125
alternate text, 126
alternative voice (italic), 87
"Anatomy of a Mobile-First Responsive Web Design" (Frost), 450
ancestors, 215
anchor element. *See* a element
Android Browser, 18
animation
  animatable CSS properties, 401
  canvas element used for, 199
  CSS animation properties, 422
  GIF images, 514
  keyframe, 420–423
  SVG images, 537
  tools for, 421
anti-aliasing, 531
Apache servers, 22
APIs (Application Programming Interface)
  defined in HTML5, 185, 189–192
  W3C and WHATWG, for use with web applications, 190
appendChild( ) method, 491
arbitrary substring attribute value selector, 282
arc( ) function, 201
arguments, 473, 475
ARIA (Accessible Rich Internet Applications), 42
arrays, 468
article element, 79, 81
artifacts, 516
aside element, 79, 81, 82
aspect ratio, preserving, 447
ASP.NET (Active Server Pages), 9, 150
assistive technologies, 23
Asynchronous JavaScript and XML. *See* Ajax
Ateş, Faruk, 495
attribute nodes (DOM), 486
attributes, 61
  changing values of, using JavaScript, 489
  global, 187
  specifying event handlers to run, 479
  syntax for, 62
  XHMTL requirements for, 183
attribute selectors, 281–283
Audacity video and audio converters, 194
audio and video (HTML5), 192–198
  adding audio to a page, 198
  audio formats, 194
  media formats, how they work, 192
audio element, attributes, 198
authoring, 8
  links to popular tools, 19
  progressive enhancement strategy, 37
author style sheets, 217
autoplay attribute, video element, 196

B

backend development, 9
background
  color, 273
  images, 124, 284–292
  multiple backgrounds, 294–295
  shorthand background property, 293–295
  watching for overrides, 293
baseline profile (H.264 codec), 197
BBEdit, 17
bdi (bidirectional isolation) element, 93
bdo (bidirectional override) element, 93
:before and :after pseudo-elements, 280
beginning substring attribute value selector, 282
beginPath( ) function, 201
behavior layer, 13
Berners-Lee, Tim, 90, 182
"Better CSS Font Stacks" (Nathan Ford), 231
Beverloo, Peter, 299
Bezier curves, 403
binary transparency, 526
  difference from alpha transparency, 531
bit depth, 544
bitmapped images, 522
BlackBerry, 120
BlackBerry Browser (RIM), 18
block elements, 70
  in forms, 149
blockquote element, 76
Blogger, 4
blogs, starting, 4
blurring or smoothing images, 550
BMP images, 123, 510
body element, 56, 68
  adding margin to, 329
  applying background-color property to, 274
  CSS style sheet properties applied to, inheritance of, 216
bold text
  font-weight property, 239
  strong text elements, 87
Boolean attributes, 62
Boolean values, 467
Boot Camp, 15
border-box model, 309
border-collapse property for a table, 142
borders, 306, 316–327
  animatable CSS properties, 401
  color, 319
  combining style, width, and color, 320
  outlines versus, 380
  rounded corners with border-radius, 321
  style properties, 316
  table, separated and collapsed, 441–443
  using border-image property, 323–327
  width (thickness), 318

Obras protegidas por Direitos de Autor

border-spacing property for a table, 142
bottom property, 357
Bowman, Doug, 397
box model (CSS), 219, 305
  specifying box dimensions, 306–312
box-shadow property, 335
box-sizing polyfill script, 309
box-sizing property, 307
Bradshaw, Rich, 420
breakpoints, 452
br (line break) element, 64, 92
browsers
  browser wars, 493
  bugs in CSS layout, 342
  as clients of servers, 23
  CSS support, 207
  dealing with unknown capabilities, using progressive enhancement, 36–38
  default styles from built-in style sheets, 29, 60
  displaying fonts, 227
  finding pixel dimensions of images, 128
  HTML5 support, 181
    audio and video elements, 192
    Canvas, 199
  on mobile devices, 15
  parsing of HTML by, 28
  polyfills for, 493–497
  source document information ignored by, 54
  style sheet overriding styles set on body element, 216
  variety of different browsers to test pages, 17
  vendor prefixes, 298
  window object in JavaScript, 478
Budd, Andy, 330
Building Accessible Websites (Clark), 127
bullet lists, 73, 259
bullets, changing for lists, 75
button elements, 158
buttons
  button elements, 176
  radio and checkbox, 152, 161
  styling button inputs on forms, 435
  submit and reset, 152–153, 440
b (visually emphasized) element, 87

C

caching, using with images, 126
Cagle, Kurt, 538
calendar items (hCalendar), 97
calendars, 133
Calzadilla, Anthony, 420
Camen, Kroc, 197
Can I Use site's SVG listing, 537
Canvas API, 191, 198
  drawing with JavaScript, 200–204
  SVG versus, 537
canvas, background color, 273
canvas element, 11, 198
  adding to a page, 200
  examples of use, 199
capitalization, changing, 252
caption element, 143
cascade (CSS), 217
Cascading Style Sheets. *See* CSS
Cascading Style Sheets: The Definitive Guide (Meyer), 221, 247
case-sensitivity
  HTML element names not case sensitive, 55
  in filenames, 53
  JavaScript, 464
Catlin, Hampton, 433
Cederholm, Dan, 222, 410
cells
  connecting table cells and headers, 143
  empty cells, 443
  row spans, 141
  space in and between, 142
  spanning cells in tables, 139–142
  table content in, 136
  table elements for, 135
  th (table header) elements, 138
CERN particle physics lab, 90
character encoding, 56
character references, 99
checkbox buttons, 161, 162
Chikuyonok, Sergey, 553
child elements, 215
child selectors, 245
Chisholm, Wendy, 42
Chrome, 23
  developer tool, using to observe HTTP requests, 44
  validator in developer tools, 67
  vendor prefix, 298
citations, cite element, 90
Clarke, Andy, 247
Clark, Joe, 127
Clark, Keith, 496
class attribute, 97, 98
  accessing elements in DOM by, 488
  values for, 96
class selectors, 246, 276
clearing floated elements, 347
"Clever PNG Optimization Techniques" (Chikuyonok), 553
clients, 23
client-side applications, 23
clip art, 509
closePath( ) function, 201
closing (or end) tag, 55
  problems caused by omitting / (slash), 66
CMYK mode, 520
CNET's Download.com, 14
Coda by Panic, 17
codecs, 192
code element, 90
Code Style's Web Font Survey and Font Stack Builder, 231
col element, 136, 137, 146
colgroup element, 136, 137, 146
collapsed borders, for tables, 441
collapsing margins, 330
color, 265
  adding color to a document, 283
  animatable CSS properties, 401
  borders, 319
  changing for text, 243
  color names
    , 266
  gradients, 296–300
  HSL color
    , 269
  reducing number of colors in GIFs, 544
  RGBa color
    , 271
  RGB color values
    , 268
  specifying values in CSS, 265–272
  tips for using color, 274
  unpredictable color in JPEGs, 548
ColorPicker.com, 268
color pickers, 268
color selector (HTML5), 169
color table (or color map), 512

Obras protegidas por Direitos de Autor

Colorzilla, Ultimate CSS Gradient Generator, 299
cols attribute, textarea element, 155
colspan attribute, td or th element, 140
columns
  associating table header with, 144
  column-related elements in HTML5, 136
  column spans, 140
  group elements for tables, 137
  multicolumn layouts using floats, 381–391
  positioned layout, 392–394
combinators, 245
"The Coming Zombie Apocalypse", 35
comments, 54
  adding to HTML documents, 60
  in CSS style sheets, 213
  JavaScript, 465
comparision operators, 468
"Comparison of JavaScript frameworks", 498
Complete Guide to Pre-Installed Fonts in Linux, Mac, and Windows, 231
compression
  GIF, 513
  JPEG, 515
  lossy compression of JPEG images, 516
compression, JPEG, 547
concatenation of strings, 467
conditional comments, 271
conditional statements, 470
Connor, Joshue O, 42
constant-width fonts, 77, 229
  code, sample, and keyboard elements in, 90
contact information (hCard), 97
container elements, 55
container format for media, 193
containing blocks, 360–362
containing floats
  container elements, 351–353
content
  creating for HTML document, 53–54
    entering content, 53
  describing table content, 143
  empty elements not having content, 55
  organizing page content, 79–84
content area, 306
  sizing content box, 307
content management systems, 13
content strategy and creation, 10
contextual selectors, 244
controls attribute, video element, 195
copyright symbol, 99
Corel Paint Shop Pro, 508
Corel Paint Shop Pro Photo, 17, 268
Coyier, Chris, 134, 222, 299
createElement( ) method, 490
createTextNode( ) method, 491
"Creating Accessible Images" at WebAIM, 127
"Creating Accessible Tables" at WebAIM, 144
"Creating a Mobile-First Responsive Web Design" (Frost), 450
"Creating Intrinsic Ratios for Video" (Koblentz), 447
Creative Commons licenses, 509
Crockford, Douglas, 483
cryptography, public-key, 170
.css file extension, 301
"CSS Specificity: Things You Should Know", 247
"CSS: Specificity Wars" (Clarke), 247
CSS (Cascading Style Sheets), 8, 12, 207–224
  adding style sheets to simple web page, 64
  benefits of using, 207–209
  changing unordered list appearance, 74
  color and background properties review, 304
  color property, using to change text color, 243
  color values, specifying, 265–272
  comments in style sheets, 213
  conflicting styles, the cascade, 217–219
    assigning importance to style rules, 218
    style sheet hierarchy, 219
  external style sheets, 300–303
  future of CSS layout, 381
  inheritance, 214–217
    document structure and, 215
    of certain style sheet properites, 216
    parent and child elements, 215
  resources for further information, 221
  responsive web design with, 38–40
  selectors, 244–248, 496, 583–586
  style sheets, how they work, 209–214
    attaching styles to document, 213
    marking up the document, 209
    writing tules, 210
  styling web pages with, 29
  techniques, 427–456
    CSS Reset, 427–428
    image replacement, 429–430
    responsive web design, 444–454
    sprites, 430–433
    styling forms, 434–441
    styling tables, 441–443
  tools, 222
  units of measurement, 234
CSS Cookbook (Schmitt), 222
CSS-discuss Wiki, 342
CSS for Web Designers (Cederholm), 410
CSS Sprite Generator, 432
CSS: The Missing Manual (McFarland), 221
CSS-tricks (blog), 222
CSS Zen Garden, 208, 378
cubic-bezier( ) timing function, 403
cursive fonts, 230
custom functions, 474
custom input buttons, 158
Cygwin, Linux emulator for Windows, 18

D

databases, 13
data element, 92
datalist element, 157, 176
data types, 466–468
date and time controls (HTML5), 167
dates and times, 91
datetime attribute, time element, 91
dd (definition description) element, 75
declaration block, 211
declarations
  CSS, 210
decompression, JPEG images, 517
dedicated mobile sites, 40
definition lists, 75
The Definitive Guide to HTML5 Video (Pfeiffer), 193
deleted text (del element), 92

Obras protegidas por Direitos de Autor

deprecated elements, 86
descendant selectors, 244
description lists, 75
Designing with Progressive Enhancement (Parker et al.), 38
Designing with Web Standards (Zeldman), 36
desktop browsers, 23
devices accessing the Web, 3
  mobile devices, 15
  multitude of, 34–36
device width, 445
dfn (defining term) element, 90
  background color, 274
DHTML (Dynamic HTML), 13
directory path and names, 24
disabled attribute, form control elements, 155
display property, 333
  none value for, 334
  table display values, 441
display roles, 333
disposal method, 514
dithering, reducing, 545
div elements, 95
  class attribute, 98
  id attribute, 96
dl (description list) element, 75
DNS (Domain Name System), 22
DOCTYPE declarations, 56, 311
  checking, 67
  HTML5, 185
  listing of DOCTYPEs in common use, 186
"Document Outlines" (Robinson), 81
document fragments
  linking to fragment in another document, 118
  linking to fragment within a web page, 116–117
document object
  getElementById( ) method, 488
  getElementsByClassName( ) method, 488
  getElementsByTagName( ) method, 487
  querySelectorAll( ) method, 488
Document Object Model. *See* DOM; DOM scripting
document outline, 70
document type, declarations. *See* DOCTYPE declarations
Dojo, 498
domain names, 22
  in URLs, 24
DOM (Document Object Model), 13, 58, 485–493
  accessing nodes in, 487–489
    by element name, 487
  adding and removing elements, 490–493
  browser wars and, 493
  node tree, 486
DOM scripting, 13
DOM Scripting (Keith and Sambells), 493
dots per inch (dpi), 523
drag and drop API, 191
drawing tools, 17
Dreamweaver, 16
  writing CSS style sheets, 222
drop-down menus, styling on forms, 435
dt (definition term) element, 75
DTDs (Document Type Definitions)
  HTML4.01 and XHTML 1.0 and 1.1, 185
  HTML5 and, 185
Dunham, Ethan, 228
dynamically generated pages, 13
Dynamic Drive Online Image Optimizer, 543
Dynamic HTML (DHTML), 13
dynamic programming languages, 459, 460
dynamic pseudo-classes, 276

E

ease-in-out timing function, 403
ease-in timing function, 402
ease-out timing function, 403
ease timing function, 402
East Asian languages, Ruby annotations for, 93
ECMAScript, 460. *See* JavaScript
editing API, 191
Eich, Brendan, 460
"8 Definitive Font Stacks", 231
Elastic Lawn design, 378
elastic layouts, 373, 377–379
element box, 305–306
  adding drop shadows to boxes, 335–336
  basic box properties review, 338
  borders, 316–327
  display property and display roles, 333
  margins, 328–333
  maximum and minimum dimensions, 309
  measurement properties, animatable, 401
  padding, 312–316
  specifying box dimensions, 306–312
    border-box model, 309
    handling overflow, 311
    height, 311
    sizing content box (default), 307
element nodes (DOM), 486
elements, HTML, 28
  attributes of, 61
  consisting of both content and markup, 55
  giving structure to documents, 68
  nesting, 89
  new elements in HTML5, 187
  obsolete 4.01 elements, 189
  syntax of, 55
element type selectors, 211, 276
else clause, if/else statements, 471
email
  email input type, 156
  linking to email address, 119
Embedded Open Type (EOT) fonts, 228
embedded style sheets, 213
  in style sheet hierarchy, 218
embedding video
  code example for all users, 196
embed element, 196
em (emphasis) element, 59, 86
  as inline element, 60
em measurements, 234
  font sizes, 235
  in media queries, 452
  mixing with percentages, 356
  rem (root em) measurement, 237
  using in elastic layouts, 379
empty elements, 28, 55, 61
"The Ems have it: Proportional Media Queries FTW!" (Gardner), 453
encoding of form data, 148
  specifying encoding type with file selection control, 166
encodings, 192
  video and audio encoding tools, 194

Obras protegidas por Direitos de Autor

ending substring attribute value selector, 282
end tag. *See* closing tag
EPS images, 123, 510
escaping special characters, 99
Even Faster Web Sites (Souders), 45
event handlers, 478
events, 478–480
exact attribute selector, 282
Exclusions (CSS), 381
explicit animation, 420
explicit association, form labels, 171
extended color names (X11 color names), 266
eXtensible HTML. *See* XHTML
eXtensible Markup Language. *See* XML
external links, 107
external scripts, 463
external style sheets, 213, 300–303
  attaching, using link element, 301
  importing, using @import rule, 302
  in style sheet hierarchy, 218
  modular, 302
extranets, 23

F

fantasy fonts, 230
Farkas, Alexander, 495
feature detection, 495
fieldset element, 172
fieldsets, styling on forms, 435, 439
figcaption element, 78
  for longer table descriptions, 143
figcaption elements
  long descriptions of images, 127
figure element, 78
filenames, 109
file selection controls, 165
files, naming conventions, 53
file-transfer program (FTP), 18
fill( ) function, 202
fillStyle attribute, Canvas API, 202
fillText( ) function, 202
Firebug plug-in for Firefox, 67
Firefogg, 194
Firefox, 23
firewalls, 23
Fireworks, 17, 268, 508
  converting RGB to indexed color, 512
  creating animated GIFs, 514
  JPEG optimization options, 547
  making a color transparent, 533
  Matte color feature, 532
  Optimize and Preview panels, 542
  Optimize panel, 528
  Optimize to File Size feature, 553–554
  preserving original quality of JPEGs, 516
  quality (compression) rates, 549
  resizing image using Image Size dialog, 526
  saving images in chosen format, 521
  Selective Quality, 550
  smooting images, 550
first-letter pseudo-element selectors, 280
first-line pseudo-element selectors, 280
fixed layouts, 373, 374–375
  three columns, 386–387
  two columns, 382, 384
    centered, 385
fixed positioning, 357, 368–369
"Fixed Positioning in Mobile Browsers" (Frost), 369
Flanagan, David, 463
Flash, 192
  advantages and limitations of, 11
  fallback for browsers not supporting HTML5 video, 196, 197
  phasing out of, 192
FlashCanvas JavaScript library, 199
Flexbox (Flexible Box Layout Model), 381
flexible images, 445, 447
Flickr, Creative Commons, 509
floating, 342–356
"Fluid Grids" (Marcotte), 376
fluid layouts, 373, 376, 445
  faux columns for, 396
  using max-width property, 383
focus state, 277
Fontdeck, 229
font embedding services, 228
font-face property, 216
Font Matrix website, 231
fonts
  animatable CSS properties, 401
  constant-width, 77
  CSS properties for, 225–242
    font-family property, 226–232
    font property, 241
    font-size property, 233–238
    font-style property, 240
    font-variant (small caps), 241
    font-weight property, 239
  font attribute, Canvas API, 202
  web fonts, 228
Fonts.com, 229
Fontspring, 228
font stack, 226
  strategies for, 231
font-style property, 88
font-weight property, 87
footer element, 79, 83
Ford, Nathan, 231
for loops, 471
form controls, 147, 149, 152–170
  adding radio buttons and checkboxes, 162
  association with labels, 171
  color selector (HTML5), 169
  date and time controls (HTML5), 167
  disabled and readonly attributes, 155
  file selection, 165
  hidden, 166
  menus, 163
    adding a menu, 165
    grouping menu options, 164
    pull-down menus, 163
    scrolling menus, 163
  name attribute, 151
  numerical inputs (HTML5), 168
  radio and checkbox buttons, 161
  submit and reset buttons, 158–160
  text entry controls, 153–158
forms, 147–180
  accessibility features, 171–173
  element review, 176–180
  form controls, 152–170
  form element, 149–151, 176
    action attribute, 149
    method attribute, 150
  how they work, 147–148
  layout and design, 173–175
  new form elements in HTML5, 170
  new input types in HTML5, 187
  styling, 434–441
  variables and content, 151–152
    name attribute, 151
foundries, 228
fragment identifiers, 116

Obras protegidas por Direitos de Autor

fragment, linking to
in another document, 118
within a web page, 116–117
frame delay, 514
frames, 514
Frameset DTD, 185
frontend versus backend of website creation, 9
Frost, Brad, 34, 450, 369
FTP programs, 18
links to popular software, 19
Fulton, Steve and Jeff, 202
functional notation, 284
functions, 473–476
arguments, 473, 475
custom, 474
native JavaScript functions, 474
returning a value, 475
variable scope and, 476
Future Friendly website, 36

G

games, canvas element used for, 199
gamma, 518
Gardner, Lyza Danger, 40, 453
general sibling selectors, 245
generated content, 280
generic elements, 95–98
div, 95
span, 96
generic font families, 229–230
Geolocation API, 191
getElementById( ) method, 488
getElementsByClassName( ) method, 488
getElementsByTagName( ) method, 487
GET method, 151
Getty Images, 509
GIF images, 123, 511–515, 517
8-bit indexed color, 511
animation, 514
best uses of, 511
compression, 513
designing for compression, 542
interlacing, 514
making transparent GIFs, 528–533
optimizing, 543–547
transparency, 513
GIMP (GNU Image Manipulation Program), 17, 508
GitHub, 309
global attributes, 187
globally scoped, 476
global variables
problems with, 477
Google
Developers site, 498
Let's Make the Web Faster site, 45
Web Fonts service, 229
government accessibility requirements (Section 508), 42
gradients, 272, 296–300
graphical browsers, 23
Graphic Interchange Format. *See* GIF images
graphics. *See also* images; web graphics
SVG (Scalable Vector Graphics), 533–538
web-graphics-creation tools, 17
graphics tools, 508
grayscale, 509
PNG support of, 518
Greenwich Mean Time (GMT), 91
grids
CSS grid frameworks, 375
grid layout system, 381
Griffiths, Patrick, 378
Grigsby, Jason, 40
grouped selectors, 220, 276
grouping related inputs, 174
"The Guide to CSS Animation: Principles and Examples", 420
Gustafson, Aaron, 38, 158, 396, 430

H

h1 (Heading Level 1) element, 59, 70
h2 (Heading Level 2) element, 59, 71
H.264 video codec, 193
profiles, 197
halos, 531
Handbrake encoding tool, 194
Handcrafted CSS (Cederholm), 222
hanging indents, 251
Hay, Stephen, 35, 381
head element, 68,
header element, 79, 83
headers attribute, td element, 144
headers, tables, 138–139
connecting table cells and headers, 143
Head First Mobile Web (Gardner and Grigsby), 40
headings, 70–73
hero images, 524
hexadecimal (hex), 269
RGB values, 270
hgroup (headings group) element, 73
Hickson, Ian, 184
hidden controls, 166
High Performance Web Sites (Souders), 45
horizontal alignment of text, 251
hosting plans, 150
hosting services for blogs, 4
hover state, 277
Hawryluk, Zoltan "Du Lac", 336
"How to Simulate CSS3 box-shadow in IE6-8 without JavaScript" (Hawryluk), 336
hr (horizontal ruled line) element, 28
href attribute, a (anchor) element, 106–107
hr (horizontal ruled line) element, 72
HSL color, 269
HTML, 8, 12, 26
authoring strategy, using progressive enhancement, 37
brief history of, 182
DOCTYPEs, 186
introduction to markup, 26
invalid markup, problems caused by, 65–67
learning by typing it manually, 50
minimizing documents, 43
obsolete HTML 4.01 text elements, 86
version 4.01, 183
versions of, 26
HTML5, 12, 69, 181–204
APIs, 189–192
browser support of, 181
markup, 185–189
new form elements, 170
online validator for documents, 67
support in Internet Explorer, 80
video and audio, 192–198
"HTML5 Sectioning Elements, Headings, and Document Outlines" (Johansson), 81
HTML5 Canvas (Fulton), 202
HTML5 for Publishers (Kleinfeld), 200

Obras protegidas por Direitos de Autor

HTML5 Graphics with SVG & CSS (Cagle), 538
HTML5 Media (Powers), 193
HTML5 shiv (or shim), 494
HTML5, Up and Running (Pilgrim), 193
HTML documents, 26
  creating in Notepad, 51
  creating in TextEdit, 52
  elements giving structure to, 68
  validating, 66–67
HTML editors, 16
html element, 68, , 359
HTTP (HyperText Transfer Protocol), 21, 23
http:// portion of URLs, 24
HTTP requests, reducing number of, 43, 430
https:// in URLs, 24
hue, saturation, and lightness. *See* HSL color
hybrid layouts, 373, 379
hypertext links, 21. *See also* links
HyperText Markup Language. *See* HTML
hyphen-separated attribute value selector, 282

I

Icon Finder, 510
icons sources of, 510
id attribute, 245, 97–99
  accessing elements in DOM by, 488
  using to name element as fragment identifier, 116
i (different mood or voice) element, 87, 88
Idiomatic Style Manifesto (for JavaScript), 470
ID selectors, 245, 276, 501
IE. *See* Internet Explorer
if/else statements, 470, 481
IIS (Internet Information Services), 22, 150
illustrations, 508
  SVG format for line-style illustrations, 533
Illustrator, 17, 508
image editing and drawing software, links to, 19
image-editing tools, 268
images, 123–132
  accessibility of, 127
  adding and linking (exercise), 129
  adding to a web page, 61–64
  background, 284–292
  editing in RGB mode, 520
  file formats, 123
  flexible, 445, 447
  floating, 348
  formats, 510–522
    choosing best format, 519
    file extensions, 511
    GIF, 511–515
    JPEG, 515–517
    PNG, 517–519
    saving image in chosen format, 520–522
  image buttons, 158
  img elements, 124–130
  optimizing, 43
    general strategies for, 542–543
  PNG-8 alpha transparency, 531
  replacement techniques, CSS-based, 429–430
  resizing on the server, 453
  responsive, for mobile devices, 453
  reusing and recycling, 542
  size and resolution, 522–526
  sources of, 507–510
  SVG (Scalable Vector Graphics), 533–538
  using border-image property, 323–327
  using caching with, 126
  working with transparency, 526–533
img (image) element, 61, 123, 124–130
  adding to example web page, 63
  alt attribute, 126
  floating, 342
  required attributes, 63, 125
  src attribute, 61, 115, 125
  width and height attributes, 128–132
implicit animation, 420
implicit association, form labels, 171
important text (strong element), 86
incorrect text (s element), 88
indentation
  in HTML source, 100
  text-indent property, 250
indexed array members, 468
indexed color, 512
indexed color images
  palettes for, 512
  PNG images, 517
index file, 25
  creating and examining in a browser, 54
information architect (IA), 10
information designer. *See* information architect
inheritance
  CSS style sheet properties, 214–217, 227
    inherits keyword, 239
initial containing block, 359
inkscape.org, 535
inline elements, 60, 84–94
  applying background-color property to, 274
  floating, 344
  in normal page flow, 341
  margins on, 330
inline style, applying, 214
innerHTML, 487, 490
input element, 153, 177
  type attribute
    available attributes for each input type, 179
  button, 158
  checkbox, 162
  color, 169
  date-and time-related (HTML5), 167
  email, tel, url, and search types, 156
  file, 165
  hidden, 166
  image, 158
  new form input types in HTML5, 187
  number and range, 168
  password, 155
  radio, 161
  reset, 158
  submit, 158
  text, 154
inputs, aligning on a form, 438
insertBefore( ) method, 492
inserted text (ins element), 92
interaction design (IxD), 5
interlacing, 514
  PNG images, 518
Internet Explorer, 23, 182. *See also* browsers

Obras protegidas por Direitos de Autor

audio support, 194
border images, no support for, 323
border-radius property and, 322
box model bug, 311
box-shadow property and, 336
box-sizing property and, 309
browser wars, 493
bugs in CSS layout, 342
Canvas support, 199
conditional comments, 272
Embedded Open Type (EOT) fonts, 228
hgroup element support, 73
JavsScript script creating HTML5 elements for, 80
media queries and, 449, 451
no text-zoom on type sized in pixels, 236
RGBa color support, 271
text shadows not supported, 254
vendor prefix, 298
video support, 193
Internet Explorer Mobile, 18
intranets, 23
iPad, 34
Retina displays, saving JPEGs as Progressive for, 550
IP addresses, 22
iPhone, 34
detecting phone numbers and turning them into links, 120
IP (Internet Protocol), 22
Irish, Paul, 228, 495
iStockPhoto, 509
italic text, 87, 88
citations, 90
font-style property, 240
variables in, 90

J

JavaScript, 9, 13, 459–484
accessing DOM nodes, 487–489
adding behaviors to web pages, 29
adding to a page, 463
anatomy of a script, 463–477
basic syntactical rules, 464
built-in functions, 464
comments, 465
comparison operators, 468
functions, 473–476
if/else statements, 470
loops, 471–473
variables, 465–468
variables scope and var keyword, 476
capabilities of, 461
defined, 460
drawing with, using Canvas API, 200–204
events, 478–480
libraries, 497–501
how to use jQuery, 499–501
SVGWeb JavaScript library, 537
minimizing, 43
misconceptions about, 460
polyfills for browsers, 493–497
progressive enhancement strategy for, 37
resources for further information, 483
script examples, 481–482
JavaScript & jQuery: The Missing Manual (McFarland), 483
JavaScript: The Definitive Guide (Flanagan), 463
JavaScript: The Good Parts (Crockford), 483
JavaServer Pages (.jsp), 150
Jehl, Scott, 312, 496
Jensen, Scott, 35
Johansson, Roger, 81
Joint Photographic Experts Group. *See* JPEG images
JPEG images, 123, 511, 515–517
24-bit Truecolor images, 515
designing for compression, 542
lossy compression, 516
optimizing, 547–552
progressive, 516
jQuery, 498
.js file extension, 463
JSON (JavaScript Object Notation), 497
JW Player, 197

K

kbd (keyboard) element, 90
Keith, Jeremy, 493
Kellum, Scott, 429
keyboards suited to entry tasks, 156
keyframe animation, 420–423
keygen element, 170, 177
keywords
positioning background image, 288
using to size text, 237
visual emphasis of, 87
Kindle Fire, Silk browser, 18
Kleinfeld, Sanders, 200
Koblentz, Thierry, 447
Koch, Peter-Paul, 119
Konqueror, vendor prefix, 298

L

label attribute, optgroup element, 164
label element, 171, 177
labels
aligning on a form, 438
placement on forms, impact of, 174
styling on a form, 439
languages, non-Western, accommodating, 93
layout, 373–398
custom, based on viewport size, 38–40
multicolumn layouts using floats, 381–391
page layout strategies, 373–380
positioned layout, 392–394
Learning JavaScript (Powers), 483
left property, 357
left-to-right (ltr) reading languages, 93
legal small print, 88
legend element, 172, 177
legends, styling on forms, 435
Lennartz, Sven, 431
LESS (CSS syntax), 433
letter-spacing property, 253
libraries, JavaScript, 497–501
lightness (HSL color), 269
li (list item) element
in ordered lists, 74
in unordered lists, 73
linear timing function, 402
line breaks, 54, 92
br element, 64, 92
line-height property, 249
line length, 374
lineWidth attribute, Canvas API, 202
link element
media attribute, media queries in, 449
referring to external CSS style sheets, 301

Obras protegidas por Direitos de Autor

links, 105–122
  bottom borders instead of underlines, 316
  changing color for, 105
  href attribute, 106–107
  linking to pages on the Web, 107
  mail links, 119
  primary navigation, wrapped in nav element, 82
  pseudo-classes, 276
    changing colors and backgrounds of links with, 278
  targeting new browser window, 118–119
  telephone links, 120
  writing URLs for, 121
Linux
  Cygwin, Linux emulator for Windows, 18
  web authoring tools for, 16
Liquid Bleach technique, 397
liquid layouts, 373. *See also* fluid layouts
list-item display role, 260
lists, 73–76
  changing bullets and numbers, 259–261
  description, 75
  navigation menu as unordered list, 350
  nesting, 74
  ordered, 74
  unordered, 73
list-style-type style sheet property, 75
locally scoped, 476
loop attribute, video and audio elements, 196
loops, JavaScript, 471–473
loosely typed programming languages, 459, 460
lossless compression, 513
lossy compression, 516

M

Macintosh computers. *See also* Mac OS X
  running Windows on, 15
Mac OS X
  Apache server, 22
  creating HTML document with TextEdit, 52
  FTP programs, 18
  "Mac Dec Bin Hex Calculator" for OS X, 271
  Terminal application, 18
  web authoring tools for, 16
Macromedia Flash. *See* Flash
mailto links, 119
Make the Web Faster site, 45
Marcotte, Ethan, 38, 236, 376, 444
margins, 306, 328–333
  adding to elements, 332
  animatable CSS properties, 401
  applying to body and individual elements, 329
  behavior of, 330
  CSS properties for, 328, 338
mark element, 91
markers for list items, 259–260
markup, 8
  CSS style sheets and, 209
  tags around content, 55
  unrecognized markup ignored by browsers, 54
markup languages, 12
"A Masterclass in CSS Animations" (Weyl), 420
mathematical operators, 469
MathML, 14
Matte color feature, 532
Max audio converter, 194
maxlength attribute, text input field, 154
May, Matt, 42
McFarland, David Sawyer, 221, 483
measurement units (CSS), 234
measurement values, 211
mediagroup attribute, video element, 196
media player API, 190
media queries, 445, 448–450
  adding, 450
  choosing beakpoints for, 452
  device features you can detect with, 448
  examples in style sheet, 448
  in the document head, 449
  min-width and max-width, polyfill for, 496
  mobile first, 450
Media Queries gallery site, 38, 444
media types, 448
menus
  on forms, 163
    adding a menu, 165
    grouping menu options, 164
    pull-down menus, 163
    scrolling menus, 163
metadata, 97
meta element, 56, 68
meter element, 170, 177
method attribute, form element, 150
methods, event handlers as, 480
Meyer, Eric, 221, 247, 299, 330
Microformats, 97
Microsoft
  ASP.NET (Active Server Pages), 150
  browser wars, 493
  Internet Information Services (IIS), 22, 150
  Office, 231
  vendor prefix, 298
  Windows. *See* Windows
Microsoft Expression Web, 16
min-width and max-width properties, 377, 383
Miro Video Converter, 194
mobile browsers, 18, 23
mobile devices
  calling a number on your site, 120
  hover state, 278
  mapping of reference pixel to hardware pixels, 524
  mobile-first media queries, 450
  problems with overflow property, 312
  problems with position: fixed property, 369
  requirement for testing web designs, 15
  SVG support, 538
  use of mobile phones as only means to access Internet, 35
mobile-first design approach, 450
Mobile First (Wroblewski), 35
Mobile Safari, 18
  progressive images for, 517
  viewport width for iPhones, 445
mobile sites built separately from primary sites, 39
Mobile Web, 35
Mobile Web Best Practices site, 454
mobility impairment, designing for, 41
Mobitest by Blaze, 45
Modernizr, 495

Obras protegidas por Direitos de Autor

modular style sheets, 302
Moll, Cameron, 377
monospace fonts, 77, 229
monospace font stack, 231
MooTools, 498
Morris, Trevor, 519
Mosaic Netscape, 182
Mozilla Developer Network, 480
Mozilla Foundation, vendor prefix, 298
MP3/WMA/Ogg Converter, 194
.mp4 or .m4v file suffix, 193
MPEG-4, 193, 197
multiline text entry fields, 154
multimedia, 10
multimedia players, 192
multipart/form-data encoding type, 166
multiple attribute, select element, 164
"The Mystery of CSS Sprites: Techniques, Tools, and Tutorials" (Lennartz), 431

N

naked or anonymous text, 70
name attribute, 151
  text input field, 154
named entities, 99
names, font, 226
naming conventions for files, 53
native functions, 474
nav element, 79, 81, 82
navigation
  creating a navigation bar, 350, 353
  primary navigation links, 82
negative margins, 331
  using to create any order columns, 388–391
nested elements, 57
Netscape, 182
  browser wars, 493
networking software, links to, 19
nodeList, 487
nodes, 486
  accessing in the DOM, 487–489
Nokia Series 40 and Nokia Browser for Symbian, 18
"No Margin for Error" (Budd), 330
non-breaking space ( ), 100
non-replaced elements, 125, 307
Notepad, creating HTML document on, 51
The Noun Project, 510
  SVG artwork, 533
null data type, 466
numbered lists, 74
#1 Free Clip Art, 510
numbers
  changing for lists, 75
  numbering styles for ordered lists, 260
  numeric values assigned to JavaScript variables, 466
numerical inputs (HTML5), 168
numeric entities, 99
Nvu editor, 16

O

object element, 196
  pointing to external .svg file, 536
  using to embed Flash player, 197
octothorpe symbol (#) before fragment name, 116
offline web application API, 190
offset properties, 357
  setting pixel measurements for positioned element, 363
Ogg Theora, 193
.ogv file suffix, 193
ol (ordered list) element, 74
  start attribute, 75
online image optimizers, 543
opacity, 275–276
  animatable CSS properties, 401
opening and closing HTML tags, 28, 55
open source software, 22
OpenType format (OTF) fonts, 228
Opera, 23
Opera Mobile and Mini, 18
operating systems. *See also* names of individual operating systems
optgroup elements, 163, 164, 177
Optimize to File Size feature, Photoshop and Fireworks, 553
optimizing, 541. *See also* images, optimizing
option element, 163, 177
  in scrolling menus, 164
  selected attribute, 164
Oracle VirtualBox, 15
ordered lists, 74
  changing numbers, 75
  numbering and lettering styles for, 260
Ordering Disorder, Grid Principles for Web Design (Vinh), 375
origin image, 288–290
origin point for transformations, 411
outlines, 380
  animatable CSS properties, 401
outlining system (HTML5), 70, 76
output element, 170, 178
overflow property, 311
Overthrow script, 312

P

padding, 306, 312–316
  animatable CSS properites, 402
  for cells in tables, 142
page layout with CSS. *See* layout
Painting the Web (Powers), 538
palettes, 512
paragraph-level thematic break, 72
paragraphs, 70
  thematic break, hr element, 72
parallax motion, 295
Parallels Desktop for Mac, 15
parent elements, 215
Parker, Todd, et al., 38
partial attribute value selector, 282
password input type, 155
pathnames
  dot-dot-slash (../) convention, 112
  site root relative, 114
Pederick, Chris, 222
percentage values
  background position measurements, 288
  in hybrid layouts, 379
  mixing with ems, 356
  specifying position of an element, 364
  using to create fluid layouts, 377
performance
  box shadows, text shadows, gradients and, 336
  site, 43–45
period. *See* . (dot), under Symbols
persona, 7
perspective-origin property, 419
perspective property, 419
Pfeiffer, Sylvia, 193
Phark image replacement technique, 430

Obras protegidas por Direitos de Autor

photography, stock, 508
Photoshop, 17, 268, 508
  color table for GIF images, viewing, 512
  creating animated GIFs, 514
  JPEG optimization options, 547
  making a color transparent, 532
  Matte color feature, 532
  Optimized option, 550
  Optimize to File Size feature, 553–554
  resizing image smaller, 525
  Save for Web feature, 516, 528
Photoshop Elements, 17
PHP, 9
  scripts written in, 150
phrasing content, 70
Pilgrim, Mark, 193
"A Pixel Identity Crisis" (Kellum), 524
Kellum, Scott, 524
pixel dimensions
  finding for image using browsers, 128
  specifying accurately, 128
pixels, 234, 522
  and absolute measurements, 236
  image, mapping to hardware pixels, 523
pixels per inch (ppi), 522
placeholder attribute, 155
PNG images, 123, 511, 517–519
  making transparent PNGs, 528–533
  optimizing, 552–553
  PNG-8 alpha transparency, 531
  progressive display (interlacing), 518
  transparency, 518
polyfills, 493–497
pop-up windows, 119
Portable Network Graphic. *See* PNG images
positioning, 341
  absolute, 359–368
  fixed, 368–369
  relative, 358
  specifying position, 357
  types of, 356
positioning context, 360
Position Is Everything site, 342
poster attribute, video element, 195
POST method, 150
  using with forms having a file selection control, 166
Powers, Shelley, 193, 483, 538
p (paragraph) element, 59, 70
"Pragmatic Responsive Design" (Rieger), 449
"Prefix or Posthack" (Meyer), 299
preload attribute, video element, 196
pre (preformatted text) element, 77
presentation, 12, 29
  defining with CSS, 207
  early use of HTML elements for, 58
  inline elements used for, 85
presentation layer, 13, 209
"Principles of Writing Consistent, Idiomatic JavaScript", 470
production, 8–9
program code elements, 90
programming, 9
  server-side, 13
progress element, 170, 178
progressive display
  JPEG images, 516
  PNG images, 518
progressive enhancement, 36–38, 39
Progressive format, JPEGs for iPad Retina displays, 550
Pro HTML5 Accessibility (Connor), 42
Proietti, Valerio, 498
properties CSS, listings of, 227
protocols, 21
  in URLs, 24
Prototype JavaScript Framework, 498
pseudo-class selectors, 276–279
  changing colors and backgrounds of links with, 278
  other, 279
pubdate attribute, time element, 92
public-key cryptography, 170
pull-down menus, 163
PunyPNG, 543
"Push My Button" (Gustafson), 158
PuTTY, Telnet/SSH client, 18
Python, 9

Q

q (quotation) element, 77, 89
Quartorolo, Tony, 431
querySelectorAll( ) method, 488
Quirksmode, 119
quotations in text
  lengthy quotations, 76
  short quotations, 77, 89

R

radial gradients, 297
radio buttons, 161
  labels for, 171
Raggett on HTML4, 182
range input type, 168
raster images, 522
RDF Site Summary, 14
readonly attribute, form control elements, 155
ready event (jQuery), 500
RealMedia, 192
reference pixel, 523
Regions and Exclusions (CSS), 381
relative keywords, using to size text, 237
relative (measurement) units, 234
relative pathnames, 108
relative positioning, 356, 358
relative URLs, 106
rel attribute, 97
rem (root em) measurement, 237
replaceChild( ) method, 492
replaced elements, 125
reset (CSS), 427–428
Resig, John, 498
ResizeMyBrowser.com, 452
resizing images, avoiding using HTML for, 128
Respond.js, 496
"Responsive Web Design" (Marcotte), 38, 236
"Responsive Data Tables" (Coyier), 134
responsive images, 453, 541
responsive web design, 38–40, 444–454
  flexible images, 447
  fluid layouts, 376, 446
  media queries, 448–450
  setting the viewport, 445
Responsive Web Design (Marcotte), 444
return keyword (JavaScript), 476
RGBa color, 271, 275
RGB color, 265, 268, 512
  RGB/Truecolor (24- and 48-bit), 517
  specifying values in CSS, 268
  writing RGB values in style sheets, 269
RGB mode, image-editing work in, 520
Rieger, Stephanie, 449

Obras protegidas por Direitos de Autor

right property, 357
rights-managed, 508
right-to-left (rtl) reading languages, 93
Robinson, Alex, 388
Robinson, Mike, 81
root directory, 114
root element, 68, , 359
root em (rem) measurement, 237
rotate transforms, 410, 411–412
rows, 135
  associating table header with, 144
  group elements for tables, 137
  row spans, 141
rows attribute, textarea element, 155
rowspan attribute, 141
royalty-free images, 509
RSS (Really Simple Syndication), 14
rt (ruby text) and rp (ruby phrasing) elements, 93
Ruby, 9
ruby annotation for East Asian languages, 93
Ruby on Rails, 150
rules, 210
  conflict between style rules, 218
    rule order and, 219
  writing, using Sass and LESS, 433
rule sets, 210
RWD. *See* responsive web design

S

Safari
  custom keyboards for input types, 156
  validator in developer tools, 67
  vendor prefix, 298
Sambells, Jeffrey, 493
samp element, 90
sans-serif fonts, 229
Sass ("Syntactically awesome style sheets"), 433
saturation (HSL color), 269
scale transforms, 410, 413
scanning, 508
  tips on, 509
Schmitt, Christopher, 222
scope attribute, associating table header with rows and column, 144
scope (variables), 476
screen readers, 23
screen sizes
  and challenges of designing for multitude of devices, 34
screen width, 445
script elements, 463
  src attribute, 463
scripting and programming, 9
  scripting strategy, using progressive enhancement, 37
scripts
  embedded on a page, 463
  external, 463
  optimizing loading of, 43
  placement in a document, 463
  server-side, processing forms, 147
scrolling menus (select element), 163
SCSS ("Sassy CSS"), 433
"The Search for One True Layout" (Robinson), 388
search engines
  analysis of heading levels, 70
  cataloging of HTML documents, 37
search input type, 156
Section 508 acccessibility guidelines (U. S. government), 41
section element, 79, 80
sectioning element, 81
sectioning roots, 76
Seddon, Ryan, 495
selected attribute, option element, 164
select element, 163, 178
  in scrolling menus, 164
Selective Quality (Fireworks), 550
Selectivizr, 496
select menus, styling on forms, 435
selectors, 209, 210, 244–248, 583–586
  accessing nodes in DOM by, 488
  attribute, 281–283
  class, 246
  conflicting style sheet rules and, 218
  descendant, 244
  element, 211
  grouped, 220
  ID, 245
  pseudo-class, 276–279
  pseudo-element, 279–281
  Selectivizr polyfill for older IE browsers, 496
  specificity of, 247
  universal selector (*), 246
  use of syntax in jQuery, 500
semantic markup, 58, 69
Sencha Animator, 421
Sencha.io Src, 453
separated borders, 441
serif fonts, 229
servers, 21
server-side applications, 23
server-side programming, 13
session history API, 190
setAttribute( ) method, 489
Sexton, Alex, 495
SGML (Standardized Generalized Markup Language), 183
shadows
  adding drop shadows to boxes, 335–336
  text-shadow property, 254–256
Shaefer, Christian, 309
Sharp, Remy, 80, 495
Shea, David, 208
sibling elements, 215
sidebars, aside elements, 82
Silk (Kindle Fire), 18
Silverlight media player, 192
s (incorrect text) element, 87, 88
site root relative pathnames, 114
skew transforms, 410, 414
Sloppy web tool, 45
slow connection speed simulators, 45
small caps font variant, 241
small element, 87
  HTML5 definition, 88
smaller (relative keyword), 237
smartphones, 15, 34
  dialing a number on your site, 120
Smashing Magazine, 431
  before and after pseudo-elements, tutorial on, 281
  "Clever PNG Optimization Techniques", 553
  "CSS Specificity: Things You Should Know", 247
Smush.it, 543
sockets, 191
software for web design, 16
  links to popular programs, 19
Souders, Steve, 45
source document, 26
  what browsers ignore in, 54
source element
  inside the video element, 196
  src and type attributes, 197

Obras protegidas por Direitos de Autor

spaces. *See also* whitespace
  between letters and words in text, 253
spacing cells, 142
spam, 120
span element, 95, 96
spanning cells, 139–142
special characters, 99–100
  avoiding in filenames, 53
specificity (CSS selectors), 218, 247
"Speed Up with CSS3 Gradients" (Coyier), 299
speed (site performance), 43–45
SpriteMe, 432
sprites, 430–433
src attribute, img (image) element, 61, 63, 115, 125
SSH client, PuTTY, 18
stacking order for elements, 367–368
standards, sticking with, 36
start tag, 55
statements, 464
static positioning, 356
step-end timing function, 403
step-start timing function, 403
steps( ) timing function, 403
stock photography and Illustrations, 508–509
storyboards, 7
Strict DTD, 185
strike-through text, 88
"Striking Web Sites with Font Stacks that Inspire" by Vivien, 231
strings, 467
stroke( ) function, 201
strokeRect( ) function, 201
strokeStyle attribute, Canvas API, 202
strong element, 86
structural layer, 209
structure layer, 13
style element, 65, 213
style property, 490
style rules. *See also* rules
  creating, 65
style sheets. *See* CSS
style tiles, 8
styling, 8
subdomains, 24
Sublime Text, 16
submit and reset buttons, 158
submitting forms, 148
sub (subscripts) element, 90
sup (superscripts) element, 90
"SVG Examples", 538
"An SVG Primer for Today's Browsers", 538
SVG (Scalable Vector Graphics), 14, 510, 533–538
SVGWeb JavaScript library, 537
syntax of HTML elements, 55

T

table element, 135
table-layout property, 441
tables, 133–146
  advanced table elements, 137
  CSS properties for, 456
  difficulty of using on small-screen devices, 134
  headers, 138
  minimal structure, 135–139
  spanning cells, 139–142
  styling, 441–443
  table accessibility, 142–144
tabs, 54
tags, HTML, 26
  components of, 55
target attribute, a (anchor) element, 119
target context = result formula for relative sizing tasks, 236
tbody (table body) element, 137, 146
td (table data) element, 135, 146
telephone links, 120
telephone numbers, 96
tel input type, 156
Telnet/SSH client, PuTTY, 18
Terminal application (Mac OS X), 18
terminal applications, 18
text. *See also* text fields
  animatable CSS properties, 401
  in comments, 54
  embedded in PNGs, 519
  floating inline text element, 344
  formatting with CSS, 225–264
    changing capitalization, 252
    changing color, 243
    CSS font properties, 225–242
    horizontal alignment, 251
    indents, 250
    line height, 249
    list bullets and numbers, 259–261
    shadows, 254–256
    spacing, 253
    underlines and other decorations, 252
  highlighting, 274
  marking up, 69–104
    figures, 78
    generic elements, div and span, 95–98
    headings, 70–73
    indicating shift in themes, 72
    lists, 73–76
    long quotations, 76
    organizing page content, 79–84
    paragraphs, 70
    preformatted text, 77
    special characters, 99–100
    text-level (inline) elements, 85
textarea element, 155, 178
  styling in forms, 434
text-decoration property, 88
TextEdit, creating HTML document, 52
text editors, 16, 50–52
  in-browser text editors, 191
text elements, 58–61
  block and inline elements, 59
  default styles applied by browsers, 60
  defining for example web page, 59
text entry controls, 152, 153–158
  HTML5 text entry, 156
  multiline, 154
  single-line, 154
  specialized text entry fields, 155
    password entry field, 155
text fields. *See also* text
text input form controls, styling, 434
text-level semantic elements, 84
TextMate by MacroMates, 17
text nodes (DOM), 486
  creating, 491
TextPad, 16
tfoot (table footer) element, 137, 146
thead (table header) element, 137, 146
thematic break, paragraph level, 72
Theora video codec, 193
th (table header) element, 135, 138, 146
TIFF images, 123, 510, 517
tiling background images, 285
  CSS Sprites and, 432
time element, 91
time zones, 91
timing functions, 402–403

Obras protegidas por Direitos de Autor

title element, 56, 68
titles for tables, 143
top property, 357
transform-origin property, 411
transforms, 410–420
  3-D, 418–421
  applying multiple, 415
  rotate transforms, 411
  scale transforms, 413
  skew transforms, 414
  transitioning, 416
  translate transforms, 412
Transitional DTD, 185
transitions, 399–409
  applying multiple, 406
  setting a delay, 404–405
  shorthand transition property, 406
  timing functions, 402–403
  transitioning transforms, 416
translate transforms, 410, 412
transparency, 526–533
  alpha levels, 271
  alpha or alpha channel transparency, 527
  binary transparency, 527
  making transparent GIFs and PNGs, 528–533
  opacity property, 275–276
  PNG-8 alpha transparency, 531
  PNG images, 511, 518
  transparent keyword value for border colors, 319
TRouBLe system for padding, 314
tr (table row) element, 135, 146
Truecolor, 515
  RGB/Truecolor (24- and 48-bit), 517
TrueType format (TTF) fonts, 228
Tuck, Michael, 231
Tumblr, 4
Tumult Hype, 421
tweening, 399
Twitter API, 190
type attribute, input element, 153
Typekit, from Adobe, 229

U

UI (user interface) selectors, 279
Ultimate CSS Gradient Generator, 299
ul (unordered list) element, 73, 107
"Uncollapsing Margins" (Meyer), 330
undefined data type, 466
underlined text, 88, 252
unicode-bid property, 257
Uniform Resource Identifiers. *See* URIs
Uniform Resource Locators. *See* URLs
United States government, Section 508 acccessibility guidelines, 41
units of measurement (CSS), 234
Universal Design for Web Applications (Chisholm and May), 42
universal selector (*), 246, 276
Unix
  Apache server, 22
  terminal applications, 18
unordered lists, 73
  changing appearance with style sheets, 74
URIs, URLs versus, 106
url input type, 156
url( ) notation, 302
URLs, 24
  parts of, 24
  requesting web pages with, 30
  URIs versus, 106
user action pseudo-classes, 277
user agent, 23
user agent string, 453
user agent style sheets, 60, 217, 427
user-centered design (UCD), 6
user experience (UX) design, 5
user flow charts, 7
user interface (UI) design, 5
user style sheets, 217
"Using CSS3 Transitions, Transforms and Animation" (Bradshaw), 420
u (underlined text) element, 87, 88

V

validation
  HTML documents, 66
  Opera client-side validation support, 157
validators, 67, 186
value attribute
  data element, 92
  text input field, 154
values
  for attributes, 62
  CSS declarations, 210
  CSS properties, 227
  of variables collected by forms, 151
variables
  collected by forms, 151
  JavaScript, 465–468, 472
    data types of assigned values, 466–468
    variable scope and var keyword, 476
  naming form variables, 152
var (variables) element, 90
vector graphics, 522. *See also* SVG
Veer stock photography and illustrations, 509
vendor prefixes (browsers), 298
Verdana-based font stack, 231
vertical alignment of text, 257
video and audio (HTML5), 192–198
  adding audio to a page, 198
  adding a video to a page, 195–198
  encoding tools, 194
  media formats, how they work, 192
video element
  attributes, 195
  source elements in, 196
"Video for Everybody", 197
video, preserving aspect ratio when scaling down, 447
viewport, 356
viewport meta element, 445
Vimeo, 196
Vinh, Khoi, 375
visibility property, 257
visited links, color of, 277
Visscher, Sjoerd, 494
VMFusion, 15
VM (Virtual Machine) products for Mac OS, 15
Vorbis audio codec, 193
VP8 video codec, 193

W

W3C (World Wide Web Consortium), 22, 182
  "An SVG Primer for Today's Browsers", 538
  APIs for use with web applications, 190
  CSS3 Multi-column Layout Module, 381
  CSS information, 222
  CSS Recommendation, 247
  CSS Regions and Exclusions, 381
  CSS Transforms, 410

Obras protegidas por Direitos de Autor

CSS Transitions Module, 400
differences in HTML5 and HTML 4.01, 187
fluid layout on website, 376
free online validator, 67
Generated and Replaced Content Module, 281
grid layout system, 381
HTML5 standard, 69
HTML5 Working Group, 184
standards as documented by, 36
SVG embedding options, 536
"SVG Examples", 538
WAI-ARIA (Accessible Rich Internet Applications) specification, 42
Web Accessibility Initiative (WAI), 41
Web Content Accessibility Guidelines (WCAG 2.0), 127
WAI-ARIA (Accessible Rich Internet Applications) specification, 42
WaSP. *See* Web Standards Project
waterfall charts, 44
wbr (word break) element, 93
Web Accessibility Initiative (WAI), 41
WebAIM
"Creating Accessible Images", 127
"Creating Accessible Tables", 144
WebAIM: Web Accessibility in Mind, 42
web applications, server-side, processing forms, 147, 148
web-authoring programs, 222
web-authoring tools, 16
Web Content Accessibility Guidelines (WCAG and WCAG 2.0), 41
web design, disciplines and roles in, 4
Web Developer extension, 222
web fonts, 228
Web Form Design (Wroblewski), 174
web graphics, 507–540, 541–556
general image optimization strategies, 542–543
image formats, 510–522
image sources, 507–510
online image optimizers, 543
Optimize to File Size feature, using, 553
optimizing GIFs, 543–547
optimizing JPEGs, 547–552
optimizing PNGs, 552–553
web graphics tools, using for image optimization, 542
Web Hypertext Application Technology Working Group (WHATWG), 184
WebINK font embedding service, 229
-webkit- vendor prefix, 298
WebKit browsers
border-radius support and vendor prefixes, 323
box-shadow property, vendor prefixes for, 336
WebM container, 193
Web Open Font Format (WOFF) fonts, 228
web page authoring tools, 19
WebPagetest, 45
Web Palette, 273
web-related programming languages, 9
"web-safe" fonts, 231
Web Safe Colors, 273
Web Sockets API, 191
Web Standards Project, 99, 183
Web Storage API, 191
Web Workers API, 191
Weizenbaum, Nathan, 433
Weyl, Estelle, 420
WHATWG HTML specification, 92
WHATWG (Web Hypertext Application Technology Working Group), 184
APIs for use with web applications, 190
whitespace
browser handling of, 54
preserving in text with pre element, 77
white-space property, 257
white-space property, 93
"Why Separate Mobile and Desktop Pages?" (Wroblewski), 40
wide sans-serif (Verdana-based) font stack, 231
wide serif (Georgia-based) font stack, 231
width and height attributes
canvas element, 200
video element, 195
width and height properties, 307
Wikipedia
"Comparison of JavaScript frameworks", 498
HTML Shiv, 495
window object, 478
properties and methods, 478
Windows
Cygwin, Linux emulator, 18
FTP programs, 18
hexadecimal converter, 271
Notepad, creating new HTML document, 51
running on Macintosh computers, 15
TextPad HTML editor, 16
web authoring tools for, 16
Windows Media, 192
Windows Phone, Internet Explorer Mobile, 18
wireframe diagrams, 6
WOFF (Web Open Font Format) fonts, 228
word break (wbr) element, 93
WordPress, 4
word-spacing property, 253
World Wide Web Consortium. *See* W3C
wrap attribute, textarea element, 155
Wroblewski, Luke, 35, 40, 174
WYSIWYG editors, 16

X

X11 color names, 266
XHTML (eXtensible HTML), 12, 14
attributes, explicit values required for, 62
empty elements in, 61
lowercase for all element names, 55
markup requirements, 183
meta element in version 1.0, 56
XHTML 2.0, 184
XML (eXtensible Markup Language), 14
drawing with, 534

Y

Yahoo!'s YSlow tool, 45
Yahoo! User Interface Library (YUI), 498
Young, Zebulon, 431
YouTube, 196

Z

z-axis, 367
Zeldman, Jeffrey, 36
z-index property, 367
zooming, full-page zoom, 378

Obras protegidas por Direitos de Autor