

“The Missing Manual series is simply the most intelligent and usable series of guidebooks...”

—KEVIN KELLY, CO-FOUNDER OF WIRED

Creating a Website

the missing manual®

The book that should have been in the box®

Third Edition
Covers HTML5



O'REILLY®

Matthew MacDonald

Creating a Website

the missing manual®

The book that should have been in the box®

Creating a Website

3rd Edition

the missing manual®

The book that should have been in the box®

Matthew MacDonald

O'REILLY®

Beijing | Cambridge | Farnham | Köln | Sebastopol | Tokyo

Creating a Website: The Missing Manual, Third Edition

by Matthew MacDonald

Copyright © 2011 O'Reilly Media, Inc. All rights reserved.

Printed in the United States.

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: <http://my.safaribooksonline.com>. For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Printing History:

April 2011: Third Edition.

Nutshell Handbook, the Nutshell Handbook logo, the O'Reilly logo, and "The book that should have been in the box" are registered trademarks of O'Reilly Media, Inc. *Creating a Website: The Missing Manual*, the Missing Manual logo, Pogue Press, and the Pogue Press logo are trademarks 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: 978-1-449-30172-9

[LSI]

Table of Contents

The Missing Credits	xiii
Introduction	1
Chapter 1: Preparing for the Web	7
Introducing the World Wide Web	7
Web Browsers	8
Web Servers	12
Planning a Website	13
Types of Sites	13
The Lifespan of Your Site	15
Practice Good Design	16
The Ingredients of a Website	17
Chapter 2: Creating Your First Page	19
Introducing HTML: The Language of the Web	19
Cracking Open an HTML File	20
Seeing the HTML of a Live Web Page	22
Creating an HTML File	22
HTML, XHTML, and HTML5	24
HTML Tags	26
What's in a Tag	26
Understanding Elements	27
Nesting Elements	29
The HTML Document	30
The Document Type Definition	30
The Basic Skeleton	32
Adding Content	34
Structuring Text	35
Where Are All the Pictures?	39
The 10 Most Important Elements (and a Few More)	41
Checking Your Pages for Errors	45

Chapter 3: Putting Your Page on the Web	49
How Web Hosting Works	49
Understanding the URL	50
How Browsers Analyze a URL	52
Domain Names	54
Choosing the Right Domain Name	54
Searching for a Name	56
Registering Your Name	57
Free Domain Names	61
Getting Web Space	62
Assessing Your Needs	63
Choosing Your Host	66
Free Web Hosts	71
Transferring Files	72
FTP	72
Browser-Based Uploading	75
Chapter 4: Power Tools	77
Choosing Your Tools	78
Types of Web Page Editors	79
Finding a Free Web Page Editor	80
Professional HTML Editors	84
Working with Your HTML Editor	85
Starting Out	86
Multiple Views	86
Creating a Web Page in Code View	88
Creating a Web Page in WYSIWYG View	89
Managing a Website	91
Defining a Site in Expression Web	92
Uploading a Site in Expression Web	94
Defining a Site in Dreamweaver	97
Uploading a Site in Dreamweaver	99
Chapter 5: Text Elements	103
Understanding Text and the Web	103
Logical Structure vs. Physical Formatting	104
CSS (Cascading Style Sheets)	105
HTML Elements for Basic Text	107
Paragraphs	109
Line Breaks	110
Headings	112
Horizontal Lines	113
Preformatted Text	114
Quotes	115
Divisions and Spans	117

HTML Elements for Lists	119
Ordered Lists	119
Unordered Lists	121
Definition Lists	121
Nesting Lists	122
HTML Elements for Tables	124
A Basic Table	124
Cell Spans	126
Inline Formatting	128
Text Formatting: Italics and Bold	128
A Few More Formatting Elements	129
Special Characters	130
Non-English Languages	132
Chapter 6: Style Sheets	135
Style Sheet Basics	136
The Three Types of Styles	136
The Anatomy of a Rule	138
Attaching a Style Sheet to a Page	139
Using an Internal Style Sheet	142
Using Inline Styles	143
The Cascade	144
Inheritance	145
More Powerful Selectors	147
Class Selectors	147
id Selectors	148
Colors	149
Specifying a Color	150
Finding the Right Color	151
Text Alignment and Spacing	152
Alignment	154
Spacing	155
White Space	156
Basic Fonts	157
Specifying a Font	159
Finding the Right Font	160
Font Sizes	161
Embedded Fonts	165
Web Formats for Fonts	166
Using a Font Kit	167
Using Your Own Fonts	170
Borders	171
Basic Borders	172
Making Better Borders	172
Using Borders with Tables	173

Tutorial: Building a Style Sheet	175
Creating a Well-Structured Style Sheet	175
Saving Work with the <div> Element	179
Saving Work with Contextual Selectors	180
Chapter 7: Adding Graphics	183
Understanding Images	183
The Element	184
Alternate Text	184
Picture Size	186
File Formats for Graphics	188
Putting Pictures on Colored Backgrounds	191
Images and Styles	194
Inline Images in Text	194
Borders	195
Wrapping Text Around an Image	196
Adding Captions	199
Background Images	201
Techniques with Graphics	204
Graphical Text	204
Backgrounds for Other Elements	207
Graphical Bullets in a List	208
Finding Free Art	209
Chapter 8: Linking Pages	213
Understanding the Anchor	213
Internal and External Links	214
Relative Links and Folders	217
Changing Link Colors and Underlining	222
mailto Links	224
Image Links and Image Maps	226
Adding Bookmarks	229
When Good Links Go Bad	231
Site Management	232
Link Checkers	233
Using Redirects	236
Chapter 9: Page Layout	239
The Challenge of Screen Space	239
Style-Based Layout	243
Structuring Pages with the <div> Element	243
Floating Boxes	244
Fixed Boxes	245
Creating a Layout with Multiple Columns	246
Building a Fixed-Width Layout	247
Creating a Resizable Layout	250
Maximum Width: The Safety Net	252

Stretching Column Height	254
Super-Flexible Sites: The Zen of Web Design	256
A Few More Layout Techniques	257
Layering	258
Combining Absolute and Relative Positioning	259
Sizing Tables	261
Chapter 10: Multipart Pages	267
Understanding Multipart Pages	267
Server-Side Includes	269
Page Templates	272
Understanding Page Templates	273
Creating a New Page Template	275
The Anatomy of a Page Template	276
Using a Page Template	279
Chapter 11: Introducing Your Site to the World	283
Your Website Promotion Plan	284
Making Your Site Search-Engine Friendly	285
Choose Meaningful Page Titles	285
Include a Page Description	287
Supply Alternate Text for All Your Images	288
Use Descriptive Link Text	289
Don't Try to Cheat	289
Registering with Directories	290
The Open Directory Project	290
The Yahoo Directory	293
Registering with Search Engines	294
Rising Up in the Search Rankings	296
Hiding from Search Engines	297
The Google Webmaster Tools	299
Tracking Visitors	302
Understanding Google Analytics	303
Signing Up for Google Analytics	305
Examining Your Web Traffic	307
Chapter 12: Website Promotion	315
Spreading the Word	316
Shameless Self-Promotion	316
Google Places	317
Google AdWords	319
Return Visitors	320
Transforming a Site into a Community	322
Fostering a Web Community	322
Website Community Tools	323
Email Newsletters	325
Twitter	327

Groups	329
Facebook	339
Chapter 13: Blogs.	349
Understanding Blogs	350
Popular Blogs	351
Syndication	353
Blog Hosting and Software	356
Getting Started with Blogger	357
Creating a Blog	358
Creating Formatted Posts	363
Managing a Blog	365
Tweaking Common Settings	368
Configuring Your Blogger Profile	370
Giving Your Blog a Custom Domain Name	372
Customizing Your Template	373
Managing Comments	379
Chapter 14: Making Money with Your Site	383
Money-Making the Web Way	384
Google AdSense	385
Signing Up for AdSense	387
The AdSense Interface	389
Creating an Ad	390
Placing Ads in Your Web Pages	396
Google-Powered Searches	399
Amazon Associates	403
Signing Up As an Associate	404
Generating Associate Links	405
PayPal Merchant Tools	411
Signing Up with PayPal	412
Accepting Payments	413
Building a Shopping Cart	419
Withdrawing Your Money	421
Chapter 15: JavaScript: Adding Interactivity.	423
Understanding JavaScript	423
Server-Side and Client-Side Programming	424
Scripting Languages	425
JavaScript 101	426
The <script> Element	426
Variables	431
Functions	434
External Script Files	439
Dynamic HTML	440
HTML Objects	440
Events	446

Image Rollovers	449
Collapsible Text	450
Interactive Forms	453
Scripts on the Web	459
Finding a Simple Script	460
JavaScript Libraries	462
Web Widgets	463
Chapter 16: Fancy Buttons and Menus	465
Fancy Buttons	465
Creating Your Button Pictures	467
Making a Rollover Effect with CSS	469
Picture-with-Text Buttons	472
Picture-less Buttons	474
Fancy Menus	475
Do-It-Yourself Collapsible Menus	476
Third-Party Menus	481
Chapter 17: Audio and Video.	487
Understanding Multimedia	488
Linking, Embedding, and Hosting	488
Types of Multimedia Files	489
Basic Background Music	492
The <embed> Element	492
Embedded Audio Options	494
Flash MP3 Players	495
The Premiumbeat Player: Playing a Song List	495
The Yahoo Player: Playing All Your Links	498
Flashtrek Loops	500
Video Clips	502
Preparing Video	502
Linking to and Embedding Video	504
Flash Video	505
Video in HTML5	506
Uploading Your Videos to YouTube	507
Signing Up with YouTube	508
Preparing a Video	508
Uploading a Video	509
Watching a Video	512
Appendix A: HTML Quick Reference	515
Appendix B: Useful Websites	543
Index	551

The Missing Credits

About the Author



Matthew MacDonald is a science and technology writer with well over a dozen books to his name. Spreadsheet fanatics can crunch numbers with him in *Excel 2010: The Missing Manual*. Data geeks can follow him into the dizzying world of databases with *Access 2010: The Missing Manual*. And human beings of all description can discover just how strange they really are in *Your Brain: The Missing Manual* and *Your Body: The Missing Manual*.

About the Creative Team

Peter McKie (editor) had the pleasure of working on the previous edition of this book. He lives in New York, where he researches the history of abandoned buildings and, every once in a while, sneaks into them. Email: pmckie@oreilly.com.

Holly Bauer (production editor) lives in Ye Olde Cambridge, MA. She's a production editor by day and an avid home cook, DIYer, and mid-century modern furniture design enthusiast by night/weekend. Email: holly@oreilly.com.

Linda Laflamme (proofreader) lives in New Hampshire where she is devoted to good grammar, perfect punctuation, vibrant vocabulary, and her son, Christopher. Email: lindalaflamme49@comcast.net.

Ron Strauss (indexer) is a full-time freelance indexer specializing in IT. When not working, he moonlights as a concert violist and alternative medicine health consultant. Email: rstrauss@mchsi.com.

Shelley Powers (technical reviewer) is a web developer and tech writer, currently living in St. Louis, Missouri. Her areas of interest are HTML5, JavaScript, and other web technologies.

Acknowledgments

No author could complete a book without a small army of helpful individuals. I'm deeply indebted to the whole Missing Manual team, especially my editor, Peter McKie, who kept me on track with relatively gentle prodding, and HTML-whiz tech reviewer Shelley Powers, who lent her keen insight about all things web-related. I also owe a hearty thanks to those who left their mark on the previous editions of this book, including Sarah Milstein, Peter Meyers, and tech reviewers Jim Goodenough, Rhea Howard, Mark Levitt, Tony Ruscoe, and Megan Sorensen. As always, I'm also deeply indebted to numerous others who toiled behind the scenes indexing pages, drawing figures, and proofreading the final copy.

Finally, I'd never write *any* book without the support of my parents, Nora and Paul, my extended parents, Razia and Hamid, and my wife, Faria. (I'd probably write many more without the challenges of my two lovable daughters, Maya and Brenna.) Thanks everyone!

—Matthew MacDonald

Introduction

These days, it's all but impossible to find someone who *hasn't* heard of the Internet. Companies create websites before they make business plans. Political activists skip the debates and trash-talk their opponents online. Even formerly technophobic grandmothers spend hours emailing old friends (and selling the odd family heirloom on eBay). The Internet has even changed our language: *Google* and *friend* are now verbs, for example, and *tweet* has nothing to do with birds.

As you no doubt know, you can establish a web presence in many ways. You can chat with friends through a Facebook page, share pictures with like-minded photographers on Flickr, put your home videos on YouTube, or write short diary-style blurbs on a blog hosted by a service like Blogger. But if you're ambitious enough to have picked up this book, you're after the gold standard of the Web: a bona fide website to call your own.

So what can you accomplish with a website that you can't do with email, social networking, and other web-based services? In a word: *anything*.

Is your personal website just a permanent place to stash your résumé or the hub of an e-commerce warehouse that sells personalized underpants? (Hey, it's made more than one millionaire.) The point is that a website of your own gives you the power to decide exactly what it is—and the control to change everything on a whim. If you're already using other web-based services, like YouTube and Facebook, you can make them a part of your website, as you'll learn in this book. For example, why not put the YouTube videos of your cat playing pool right inside your website, next to your personalized cat merchandise?

Of course, with great power comes great responsibility—meaning that if you decide to build your own site, it's up to you to make sure it doesn't look as hokey as a 1960s

yearbook portrait. That's where this book comes in. With this book by your side, you'll learn how to:

- **Create web pages.** HTML (HyperText Markup Language) is the language of the Web. Over the last decade, a modernized version HTML, called XHTML (*Extensible* HyperText Markup Language), gradually eclipsed HTML, and now is joined by another new version known as HTML5. In this book, you'll sort through these standards and learn how to write the most up-to-date, reliable web pages.
- **Make pages look beautiful using CSS (Cascading Style Sheets).** CSS picks up where HTML leaves off, adding formatting muscle that can transform the drab-best of sites into a family of coordinated pages that look like they were professionally designed. Best of all, once you understand the *right* way to use CSS, you'll be able to apply a new look to your entire site by tweaking just a single file.
- **Put your website online.** The world's greatest website isn't much good if no one sees it. That's why you'll learn how to choose the best web hosting company, pick a *domain name* (like www.HotToTrotHorses.com), and get your masterpiece online. Don't panic—plenty of cheap web hosting companies are ready to show off your site for pennies a day.
- **Attract visitors.** You'll learn how to make sure people can find your site using popular search engines and how to build an online community that encourages repeat visits.
- **Get rich (or at least earn some spare change).** The Web is a lynchpin of retail commerce, but even ordinary people can make money hawking their favorite merchandise (through Amazon), selling their own products (using a payment service like PayPal), or displaying ads (with Google). You'll learn how to get in on the action.
- **Pile on the frills.** Every website worth its salt has a few cool tricks. You'll learn how to dazzle visitors with cool buttons, slick menus, and other flashy elements, courtesy of JavaScript. You'll even learn how to (shudder) serenade visitors with background music.

What You Need to Get Started

This book assumes that you don't have anything more than a reasonably up-to-date computer and raw ambition. Although there are dozens of high-powered web editing programs that can help you build a website, you *don't* need one to use this book. In fact, if you use a web editor before you understand how websites work, you're liable to create more problems than you solve. That's because, as helpful as these programs are, they shield you from learning the principles of good site design—principles that can mean the difference between an attractive, easy-to-maintain web creation and a disorganized design nightmare.

Once you master the basics, you're welcome to use a fancy web-page editor like Microsoft Expression Web or Adobe Dreamweaver. In this book, you'll get an overview of how these two leading programs work, and you'll discover a few great free alternatives (in Chapter 4).

Note: Under no circumstances do you need to know anything about complex web programming technologies like Java or ASP.NET. You also don't need to know anything about databases or XML. These topics are fascinating, but insanely difficult to implement without some solid programming experience. In this book, you'll learn how to create the best possible website without becoming a programmer. (You *will*, however, learn just enough about JavaScript to use many of the free samples you can find online.)

About This Book

No one owns the Web. As a result, no one is responsible for teaching you how to use it or how to build an online home for yourself. That's where *Creating a Website: The Missing Manual* comes in. If the Web *did* have an instruction manual—one that painstakingly details the basic ingredients, time-saving tricks, and impressive embellishments every site needs—this book would be it.

Macintosh and Windows

One of the best things about the World Wide Web is that it truly is worldwide: Wherever you live, from Aruba to Zambia, the Web eagerly awaits your company. The same goes for the computer you use to develop your site. From an early-model Windows PC to the latest and greatest Mac, you can implement the tactics, tools, and tricks described in this book with pretty much whatever kind of computer you have. (Of course, a few programs favor one operating system over another, but you'll hear about these differences whenever they come up.) The good news is that this book is usable and suitable for owners of computers of all stripes.

About the Outline

This book is divided into five parts, each with several chapters:

- **Part One: Welcome to the Web.** In this part of the book, you'll start planning your website (Chapter 1). You'll learn the basics behind HTML, the language of the Web (Chapter 2); and you'll put your pages online with a reputable hosting company (Chapter 3). Finally, you'll look at how you can simplify your life by using web-page editing software (Chapter 4).
- **Part Two: Building Better Web Pages.** This section shows you how to add essentials to your pages. First, you'll learn your way around the CSS standard, which lets you specify fancy colors, fonts, and borders (Chapter 6). Next, you'll add pictures to your pages (Chapter 7) and create an entire website made of linked pages (Chapter 8). Finally, you'll master some slick layouts (Chapter 9) and learn how to standardize them throughout your site (Chapter 10).
- **Part Three: Connecting with Your Audience.** The third part of the book explains how to get your site noticed by search engines like Google (Chapter 11), and how to reel in web traffic (Chapter 12). You'll also take a look at *blogs* (short for *web logs*) and the free software that helps you create them (Chapter 13). Finally, you'll learn how to get on the path to web riches by displaying ads or selling your own products (Chapter 14).

- **Part Four: Website Frills.** Now that you can create a professional, working website, why not deck it out with fancy features like glowing buttons and pop-out menus? You won’t learn the brain-bending details of how to become a hardcore JavaScript programmer, but you’ll learn enough to use free JavaScript mini-programs in your own pages to perform basic tasks (Chapters 15 and 16). You’ll also dabble with movie clips and add an MP3 music player right inside an ordinary web page (Chapter 17).
- **Part Five: Appendixes.** At the end of this book, you’ll find two appendixes. The first gives you a quick reference for HTML. It explains the essential HTML elements and points you to the appropriate chapter of this book for more detailed discussions. The second appendix lists a pile of useful links that can help you learn more, get free stuff, and sign up for useful services. Don’t worry—you don’t need to type these web links into your browser by hand. It’s all waiting for you on the Missing CD page for this book at www.missingmanuals.com/cds/caw3.

About→These→Arrows

Throughout this book, you’ll find sentences like this one: “To remove Word formatting controls, choose Edit→Clear→Clear Formatting.” That’s shorthand for a much longer instruction that directs you to open three menus in sequence, like this: “Open the Edit menu by clicking Edit in the menu bar. In the Edit menu, click Clear to open a second menu. In *that* menu, click Clear Formatting to complete the process.” Figure I-1 shows a closer look.

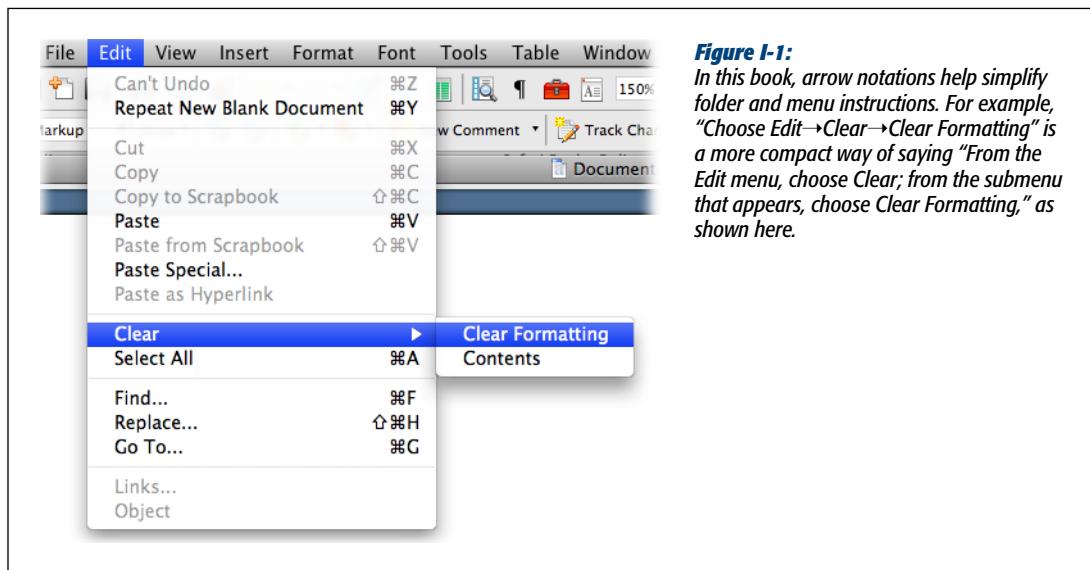


Figure I-1:

In this book, arrow notations help simplify folder and menu instructions. For example, “Choose Edit→Clear→Clear Formatting” is a more compact way of saying “From the Edit menu, choose Clear; from the submenu that appears, choose Clear Formatting,” as shown here.

About the Online Resources

As the owner of a Missing Manual, you've got more than just a book to read. Online, you'll find example files as well as tips, articles, and maybe even a video or two. You can also communicate with the Missing Manual team and tell us what you love (or hate) about the book. Head over to www.missingmanuals.com, or go directly to one of the following sections.

The Missing CD

This book doesn't have a CD pasted inside the back cover, but you're not missing out on anything. Go to www.missingmanuals.com/cds/caw3 to download examples of web page designs mentioned in this book and additional information. So that you don't wear down your fingers typing long web addresses, the Missing CD page offers a list of clickable links to the websites mentioned here.

Tip: If you're looking for a specific example, here's a quick way to find it: Look at the corresponding figure in this book. The file name is usually visible at the end of the text in the web browser's address box. For example, if you see the URL `c:\Creating a Website\Chapter 2\popsides.htm` (Figure 2-2, page 21), you'll know that the corresponding example file is `popsides.htm`.

Registration

If you register this book at oreilly.com, you'll be eligible for special offers—like discounts on future editions of *Creating a Website: The Missing Manual*. Registering takes only a few clicks. Type <http://tinyurl.com/registerbook> into your browser to hop directly to the Registration page.

Feedback

Got questions? Need more information? Fancy yourself a book reviewer? On our Feedback page, you can get expert answers to questions that come to you while reading, share your thoughts on this Missing Manual, and find groups for folks who share your interest in creating their own sites. To have your say, go to www.missingmanuals.com/feedback.

Errata

To keep this book as up to date and accurate as possible, each time we print more copies, we'll make any confirmed corrections you suggest. We also note such changes on the book's website, so you can mark important corrections into your own copy of the book, if you like. Go to <http://tinyurl.com/4ulhb8l> to report an error and view existing corrections.

Newsletter

Our free email newsletter keeps you up to date on what's happening in Missing Manual land. You can meet the authors and editors, see bonus video and book excerpts, and more. Go to <http://tinyurl.com/MMnewsletter> to sign up.

Safari® Books Online

 Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at <http://my.safaribooksonline.com>.

Preparing for the Web

The Web is an exciting place. Every day, it processes millions of financial transactions, serves up the latest news and celebrity gossip, and provides a meeting place for every type of community from political anarchists to vampire-show fans.

Since you're reading this book, you've probably decided to move in and join the Web. Congratulations! Just as you undertake some basic planning before you find a home in the real world, you need to do some preparation before you make the move to your new online neighborhood. In this chapter, you'll get a good look at how the Web works, and learn what ingredients you need to build your own website.

Introducing the World Wide Web

Although it doesn't show its age, the Internet is older than you might think. The computer visionaries who created it began developing the idea in the early 1960s. In 1969, the first transmission over the Internet took place, between a computer at the University of California at Los Angeles and one at the Stanford Research Institute. As far as pioneering moments go, it wasn't much to brag about—the computer crashed when it reached the G in the word "LOGIN." Still, the revolution was underway.

The early Internet was traveled mostly by academic and government types. It flourished as a tool for research and collaboration, letting scientists everywhere share information. In 1993, the first web browser hit the scene. In the following years, new types of people colonized the Internet, including book shoppers, news junkies, hobbyists, and a lot of lonely computer programmers.

Note: History buffs can follow the saga of the early Internet in much more detail at www.isoc.org/internet/history and www.walthowe.com/navnet/history.html.

Of course, the early Internet doesn't have much in common with today's Internet. In 1969, the Internet community consisted of four computers, all of them beastly machines that no one but a government lab or academic institution could love (or afford). In 1981, still fewer than 200 mainframe computers populated the Internet, and most of the people using them were computer experts or scientists going about their day-to-day work. Today, well over 100 million websites—and many more web enthusiasts—are online. It's no wonder you're getting so much junk email.

FREQUENTLY ASKED QUESTION

The Web vs. the Internet

Is there a difference between the Web and the Internet?

Newscasters, politicians, and regular people often use these terms interchangeably. Technically, however, the concepts are different—and confusing them is likely to put computer techies and other self-respecting nerds on edge.

The *Internet* is a network of connected computers that spans the globe. These computers are connected together to share information, but there are a number of ways to

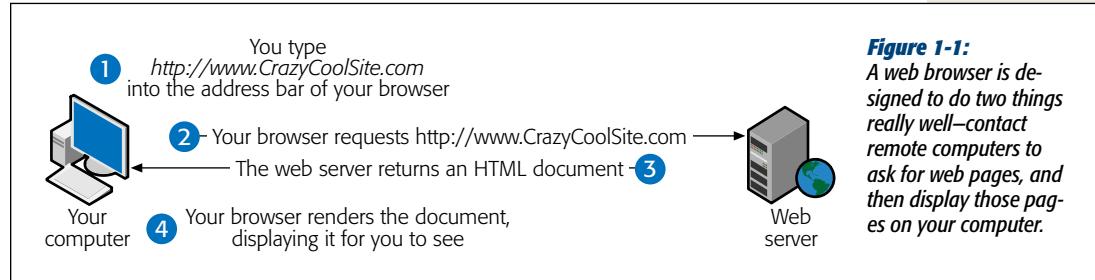
do that, including emailing, instant messaging, transferring files through *FTP* (short for File Transfer Protocol), and downloading pirated Hollywood blockbusters through peer-to-peer programs (which, of course, you don't do). The *World Wide Web* is one of the many ways to exchange information across the Internet. And how do people tap into this information? You guessed it—they use special programs called *web browsers* to visit websites and web pages spread across the globe.

Web Browsers

As you no doubt know, a web browser is a piece of software that lets you navigate to and display web pages. Without browsers, the Web would still exist, but you wouldn't be able to turn on your computer and take a look at it.

A browser's job is surprisingly simple—in fact, the bulk of its work consists of two tasks. First, it requests web pages, which happens when you type in a website address (like www.google.com) or click a link in a web page. The browser sends that request to a far-off computer called a *web server*. A server is typically much more powerful than a home computer because it needs to handle multiple browser requests at once. The server heeds these requests and sends back the contents of the desired web pages.

When the browser gets those contents, it puts its second skill into action and *renders*, or draws, the web page. Technically, this means the browser converts the plain text it receives from the server into a display document based on formatting instructions embedded in the plain-text page. The end result is a graphically rich page with different typefaces, colors, and links. Figure 1-1 illustrates the process.



Choosing your browser

Depending on your personality, choosing a web browser is either a) a bore or b) an important expression of your character, individuality, and overall computer savvy. If you fall into the latter camp, you've probably already settled on a favorite browser. But if you're searching for something a little different in a browser, or you're curious to see what else is out there, the following quick overview sums up your browser options.

Even if you're not interested in changing your browser, it's a good idea to be familiar with the most common ones out there (see Figure 1-2). That's because when you design your website, you need to prepare for a wide audience of people with different browsers. To make sure your nifty pages don't turn funky when other people look at them, it's a good idea to test your own site on other computers, using other screen sizes and other web browsers.

A Snapshot of Browser Market Share (2011)

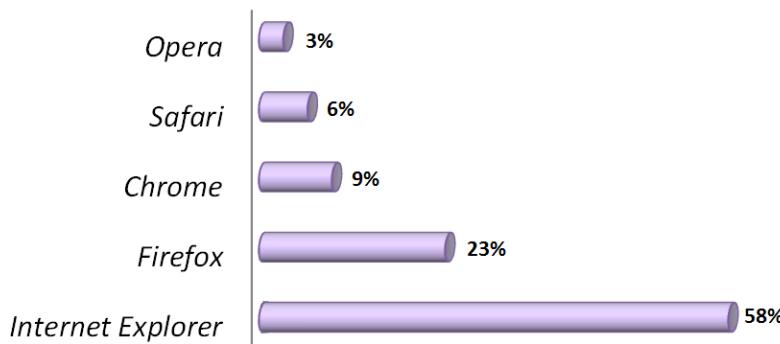


Figure 1-2:
Browser usage statistics, which estimate the percentage of people using each major browser, vary depending on what sites you examine and how you count the visitors, but at the time of this writing, this is one reasonable estimate. Just as important are browser trends, which show Firefox, Chrome, and Safari steadily creeping up in popularity at Internet Explorer's expense. (For current browser usage statistics, check out http://en.wikipedia.org/wiki/Usage_share_of_web_browsers.)

Tip: At a bare minimum, all website creators need a copy of Internet Explorer, Firefox, and Google Chrome, by far the most commonly used browsers today. By using them to check out the web pages you build, you can see what your hard work will look like to 90 percent of the world.

The following list describes the most popular browsers of today:

- **Internet Explorer** is still the world's most-used browser. For better or worse, it sets the standard that other browsers need to follow. The clear advantage of using Internet Explorer (IE for short) is that you'll never run into a web page you can't read—with a market share of near 60 percent, IE is simply too successful for site creators to ignore. The downside is that Microsoft is slow and cautious in updating IE, which means that new features often appear in other browsers first and aren't incorporated into IE until many months—or years—later.

To download an updated version of Internet Explorer, visit www.microsoft.com/ie.

Note: Although Microsoft experimented with a Mac version of Internet Explorer for years, they finally put it out to pasture when Apple created a built-in browser named Safari. So if you're a Mac owner, you'll need to borrow a friend's Windows PC to see what your pages look like in Internet Explorer. On your own computers, you'll probably use Safari or Firefox, the most popular Mac browsers.

For an extensive list of browsers for Macs, including those that work with older operating systems, see www.knutson.de/mac/www/browsers.html.

- **Firefox** started its life as the modern response to Internet Explorer. It's a web browser that's lean, secure, and more than a little hip. Firefox pioneered several innovative features long before Internet Explorer caught up, including tabbed browsing (so you can look at multiple web pages in different window "tabs") and pop-up blocking (to stop those annoying pop-up ads). Firefox is still ahead of the game with its incredibly flexible *add-ons*, tiny programs that other people develop to enhance Firefox with extra features, like a web mail notifier and thumbnails of the sites that show up in a Google search results page. Firefox currently enjoys a cult following among computer geeks and a growing number of disillusioned Internet Explorer veterans. Best of all, an army of volunteer programmers keep Firefox rigorously up to date.

Give Firefox a go at www.mozilla.org/firefox.

- **Google Chrome** is the new kid on the block. Although it didn't wink into existence until the end of 2008, its market share has already soared to third place, making it the fastest-growing browser on the planet. Although its user interface is a little rough around the edges, tech-savvy web fans love the way it runs JavaScript (the snippets of code that power interactive web pages) with blistering speed.

Experiment with Google Chrome at www.google.com/chrome.

- **Safari** is an Apple-designed browser that comes with current versions of the Mac OS X operating system. Apple products like the iPhone, iPad, and the iPod Touch use the Safari browser. It's quick, elegant, and sports a set of useful features, like spell-checking when you fill out online forms. Although Apple originally developed Safari exclusively for Mac computers, you can now download a Windows incarnation. Safari is still far more popular on Macs, however.

Go on Safari at www.apple.com/safari.

- **Opera** is a slimmed-down, easy-to-install browser that's been around for well over a decade, serving as an antidote to the bloated size and pointless frills of Internet Explorer. For years, Opera was held back by an unfortunate detail—if you wanted an ad-free version, you needed to pay. Today, Opera is free and ad-free, too, just like the other browsers on this list. It has a small but loyal following, but runs a distant fifth in web browser standings.

Check out Opera at www.opera.com.

WORD TO THE WISE

The Incredible Shrinking IE

Internet Explorer is still the world's most popular browser, and by a solid margin. But its grip on web fans is steadily loosening.

When the first edition of this book was printed (in 2005), Internet Explorer held a staggering 95 percent of the browser market. Web page designers rarely bothered to think about any other browsers. Three years later, when the second edition of this book hit the streets, IE's market share had slid to a still-respectable 80 percent. Today, Internet Explorer commands less than 60 percent of the web audience, and all signs suggest that this downward trend will continue.

Interestingly, browser-share results change depending on the web surfer's country and the type of website. For example, in Germany, Firefox is the top browser with 60 percent of web surfers. In Belarus, Opera wins the day with a 49 percent share. On the TechCrunch website (a popular news site for computer nerds), 27 percent of visitors use Firefox and another 27 percent use Chrome, while barely 16 percent use Internet Explorer.

Another variable is the type of device you use to browse the Web. The previous statistics reflect ordinary personal computer usage, but smartphones and tablets use their

own specialized browsers. For example, people using hot Apple products like the iPhone, iPad, and iPod Touch almost always do their surfing on Safari. Similarly, if Google OS (Google's operating system for netbooks and other web devices) catches on, the number people using Chrome could soar.

All these details boil down to a simple fact: Odds are, your web pages will be viewed with a variety of web browsers. So, keep these two pieces of advice in mind:

- **Stick to web standards.** To make sure everyone gets a good viewing experience, write solid, standards-based HTML and ensure that every web page includes a *doctype*, as described in Chapter 2 (page 30).
- **Test your website in different browsers.** Before you roll out a hot design for a new site, create a test page and try it out in the most popular web browsers (right now, that's IE, Firefox, and Chrome). That way, you won't unwittingly create web pages that accommodate your favorite browser's quirks. Once you lock down your initial site design, you can scale back your testing to just two browsers: IE and either Firefox or Chrome (as page rendering is usually consistent in Firefox and Chrome).

Web Servers

When you type a web address into a browser, a web server receives the browser's request and sends back the correct web page. For a busy website, this basic task can require a lot of work. As a result, web servers tend to be industrial-strength computers. Even though the average Windows PC with the right setup can host a website, it's rarely worth the effort (see the box on page 13). Instead, most people get a commercial company, called a *web host*, to give them a little space on one of their servers, usually for a monthly fee. In other words, you need to rent some space on the Web.

Sometimes, you can rent this space from the same company you use for Internet access; they may even include server space as part of your Internet connection package. But the most straightforward approach is to use a dedicated web hosting company. Either way, you're going to copy your newly built the websites to some far-off computer that will make sure a worldwide audience can enjoy your talents.

In Chapter 3, you'll learn more about how a web browser navigates the Web to find a specific page. For now, keep focusing on the big picture so you can start planning your first website.

FREQUENTLY ASKED QUESTION

Becoming a Web Host

Can I run a web server?

In theory, you definitely can. The Web was designed to be an open community, and no one is out to stop you. In practice, it's not at all easy—no matter how many computer-savvy relatives you may have.

Several monumental challenges prevent all but the most ambitious people from running their own servers. The first is that you need to have a reliable computer that runs 24 hours a day. That computer also needs to run special web hosting software that can serve up web pages when browsers request them.

The next problem is that your computer requires a special type of connection to the Internet, called a *fixed IP address*.

An IP address is a number that identifies your computer on the Web. (IP stands for Internet Protocol, which is the super-successful standard that lays down the rules that govern how different devices communicate on a network.)

In order to have your computer run a website and make sure others can find it, you need to make sure your IP address is fixed—in other words, you need to lock it down so it's not constantly changing. Most ISPs (Internet Service Providers) randomly assign new IP addresses as they're needed and change them at a whim, which means most people can't use their computers to host a permanent website—at least not without special software. If you're still interested, you can call your ISP to ask if they provide a fixed IP address service, and at what cost.

Planning a Website

The last thing you need before you start building a website is to be buried under an avalanche of theory. However, every new web author can save time and effort by doing a little bit of planning beforehand. In the following sections, you'll consider some quick guidelines to get you on the right path.

Types of Sites

You don't have much chance of creating a successful site if you haven't decided what it's for. Some people have a very specific goal in mind (like getting hired for a job or promoting a book), while others are just planning to unleash their self-expression. Either way, take a look at the following list to get a handle on the different types of websites you might want to create:

- **Personal** sites are all about you. As the world gets more web-savvy, it seems everyone is building online homes. Whether it's to share pictures of Junior with the relatives, chronicle a trip to Kuala Lumpur, or just post your latest thoughts and obsessions, a personal website is the place to do it.

When you design a personal website, it's up to you how ambitious you want to be. If all you need is a hub to chat with friends or a place to share pictures, you can save yourself a good bit of trouble by signing up on a social network like

Facebook (www.facebook.com). Or, if you want a diary-like site to chronicle your life, work, or obsessive hat-collecting hobby, a personal blog may suit you fine (see, for instance, www.blogger.com, which is discussed in Chapter 13). If you're a bit more ambitious (say you want to chart five generations of family history), or you want complete design control (forget Facebook blue), or you just need a single web home to gather together several interests, then a customized personal website is the mark of an individual with true style.

- **Résumé** sites are a specialized type of personal site and a powerful career-building tool. Rather than photocopy a suitcase full of paper résumés, why not send emails and distribute business cards that point to your online résumé? Best of all, with a little planning, your online vita can include more details than its tree-based counterpart, like links to former companies, an online portfolio, and even background music playing "YMCA" (which is definitely not recommended).
- **Topical** sites focus on a particular subject that interests you. If you're more interested in building a website about your favorite music, art, books, food, political movement, or *American Idol* contestants than you are in talking about your own life, a topical website is for you.

Before you set out to create a site, consider whether other people with a similar interest will want to visit it, and take a look at existing sites on the topic. The best topical websites attract others with the same interest. The worst websites present the same dozen links you can find anywhere else. Remember, the Web is drowning in information. The last thing it needs is another *Megan Fox Fan Emporium*.

- **Event** sites aren't designed to weather the years—instead, they revolve around a specific event. A common example is a wedding website. The event hosts create it to provide directions, background information, links to gift registries, and a few romantic photos. When the wedding is over, the site disappears—or morphs into something different (like a personal site chronicling the honeymoon). Other events you might treat in a similar way include family reunions, costume parties, or do-it-yourself protest marches.
- **Promotion** sites are ideal when you want to show off your personally produced CD or hot-off-the-presses book. They're geared to get the word out about a specific item, whether it's handmade pottery or your own software. Sometimes, these websites evolve into small-business sites, where you actually sell your wares (see the "Small business" bullet below).
- **Small business (or e-commerce)** sites show off the most successful use of the Web—selling everything from portable music players to prescription drugs. E-commerce sites are so widespread that it's hard to believe that when the Web was first created, making a buck was far from anyone's mind.

Creating a full-blown e-commerce site like Amazon.com or eBay is far beyond the abilities of a single person. These sites need the support of complex programs and computer genius-level programming languages. But if you've come to the Web to make money, don't give up hope! Innovative companies like PayPal and Yahoo provide services that can help you build shopping cart-style sites

and accept credit card payments. You can also host Google ads or hawk products from Amazon's website to rake in some cash. You'll learn about these great tricks in Chapter 14.

Once you pinpoint your website's *raison d'être*, you should have a better idea about who your visitors will be. Knowing and understanding your audience is crucial to creating an effective site. (And don't even try to suggest you're creating a site just for yourself—if you were, there's no reason to put it on the Internet at all!)

The Lifespan of Your Site

The Web constantly changes. Today's Web isn't the same as last year's—or even the Web of 15 seconds ago.

Here are two valuable truths about website lifetimes:

- **The best websites are always improving.** Their creators add support for new browser features, tweak their look to match new style trends, and—most important of all—constantly add new content.
- **When a website stops changing, it's on life support.** Many great websites have crumbled through neglect.

Think about your favorite sites. Odds are, they change daily. A good website isn't one you consult once and leave behind. It's a site you bookmark and return to periodically. In a sense, a website is like a television channel. If its creators aren't adding new information, the site is showing reruns.

Creating a website is hard enough, and keeping its content fresh is even more taxing. Here are a few tips that can help you out:

- **Think in stages.** When you put your first website online, it won't be complete. Instead, think of it as version 1, and start planning a few changes for the next version. Bit by bit, and stage by stage, you can add everything you want your site to have.
- **Select the parts you can modify regularly, and leave the rest alone.** There's no way you can review and revise an entire website every week. Instead, your best strategy is to identify sections you want to change regularly. On a personal site, for example, you might put news on a separate page and update just that page. On a small-business website, you might concentrate on the home page so you can advertise new products and upcoming specials.
- **Design a website that's easy to change.** This is the hardest principle to follow, because it requires not only planning, but a dash of hard-won experience. As you become a more experienced web author, you'll learn how to simplify your life by making your pages easier to update. One method is to separate information into several pages, so you can add new content without reorganizing your entire site. Another is to use style sheets to separate page formatting from your content (see Chapter 6). That way, you can easily insert new material without having to format the content from scratch to match the rest of your page.

Practice Good Design

Every year, hundreds of websites “win” awards for being abjectly awful. Sometimes, the profiled sites have spinning globes and hot pink text on a lime-green background. Other times, they have clunky navigation systems and grotesque flashing backgrounds. But no matter what the design sins, websites that are bad—hideously bad—are strangely common.

Maybe it’s because creating a website really isn’t that hard. Or maybe it’s because we all have an impulse to play with color, texture, and sound, and fancy web tools encourage our ugliest instincts. For a glimpse at some of the all-too-familiar mistakes, go to www.angelfire.com/super/badwebs (see Figure 1-3). You can also visit www.worstoftheweb.com, which profiles new offenders every month.



The screenshot shows a Windows Internet Explorer window displaying the "World's Worst Website". The title bar reads "World's Worst Website - Windows Internet Explorer" and the address bar shows "http://www.angelfire.com/super/badwebs/". The main content area has a yellow header with the text "The World's Worst Website" in red. Below this is a black banner with white text that says "» FIND OUT HOW VFS DIGITAL DESIGN come To My Website!". The main body of the page is orange and features several text boxes and images. One box on the left says "Gratuitous use of frames is a common mistake of web designers. Many older browsers do not support frames. They disrupt the flow of the website and can be difficult to anticipate where a page may appear when a link is clicked. Click here for an example of a frames page which is opening in the wrong window. Use your browser's 'Back' button to escape. Check out these links to websites about frames". Another box in the center says "Welcome to the World's Worst Website! This web was designed to graphically demonstrate the most common mistakes made by new Web Page designers. Where am I and where are the links to other pages? An easy to use navigation structure is essential to any well designed website! Important information should never be more than 2 clicks away." A third box at the bottom says "As you can see, this text is difficult to read. There needs to be more contrast". To the right of the page content, there is a vertical sidebar with the heading "Figure 1-3: Here's a website that gets it all wrong—deliberately. With a combination of scrolling titles, a crazily blinking background, and unreadable text, www.angelfire.com/super/badwebs does a good job of demonstrating everything you should avoid on your own web pages."

This book won’t teach you to become a professional web designer. However, it *will* help you learn the critically important art of Not Making Bad Websites. Throughout this book, you’ll find helpful tips, suggestions, and warnings about usability and design issues (look specifically for the “Design Time” boxes). In the meantime, here are a few general principles that can help make sure you never wind up on a worst-of-the-Web list (unless you absolutely want to):

- **Keep it simple (and don't annoy your visitors).** You can cram a lot of frills and goodies into a web page. But unless they serve a purpose, just say no. You'll find that exercising restraint can make a few fancy touches seem witty and sophisticated. Adding a *lot* of fancy touches can make your site seem heady and delusional. If you pare down the tricks, you'll make sure that your graphical glitz doesn't overshadow your site's content and drive your visitors away in annoyance.
- **Be consistent.** No matter how logical you think your website is, the majority of visitors probably won't think the same way. To cut down on the confusion, use similar organization from one page to another, similar headings, similar graphics and links, a single navigation bar, and so on. These touches help make visitors feel right at home.
- **Know your audience.** Every type of site has its own unwritten conventions. You don't need to follow the same design in an e-commerce store as you do on a promotional page for an experimental electric harmonica band. To help decide what is and isn't suitable, check out lots of other sites that deal with the same sort of material as yours.

UP TO SPEED

Reaching As Many People As Possible

Not only do you need to understand your audience, you also need to understand that audience's computer capabilities. Good web designers avoid using frills unless *everyone* can experience them. Nothing is more disappointing than visiting a site that's supposed to have a nice animation and seeing a blank box, simply because your computer doesn't have the right browser plug-in (an add-on program that gives your browser more capabilities). Nor is it any fun finding a website that's all decked out for wide-screen monitors, but unbearably cramped (or even worse, partly amputated) on the tiny screen of a netbook.

The creators of the most popular websites carefully consider these sorts of issues. For example, think about the

number of people whose computers won't let them buy a book on Amazon.com, make a bid on eBay, or conduct a search on Google. (Are you thinking of a number that's close to zero?) To make your website as accessible as these top sites, you need to stick to widely accepted web standards, follow the advice in this book, and try your site on different computers.

It's been widely remarked that the average web designer goes through three stages of maturity: 1) "I'm just learning, so I'll keep it simple"; 2) "I'm a web guru, and I'll prove it by piling on the features"; 3) "I've been burned by browser compatibility problems, so I'll keep it simple."

The Ingredients of a Website

Now that you know what sort of website you want to build, it's time to get practical. Building a website isn't just about picking a good design and writing solid content. You also need to coordinate with other companies to get your website onto the World Wide Web (and give it a catchy address, like www.StylinViolins.com).

This quick shopping list maps out what every website needs—and tells you where you'll learn about it in the rest of this book:

- **Web pages.** Every website is made up of individual pages. To create a basic web page, you need to understand HTML (HyperText Markup Language), the language of the Web. You'll create your first web page in the next chapter.
- **Web space.** Creating web pages is fun, but for other people to see them, you need to put them on a web server. In Chapter 3, you'll consider your options for getting your first web page online, either through a fee-based service or a free alternative.
- **A domain name.** There's a world of difference between the website address `www.inetConnections.com/Users/~jMallone012/web` and `www.JackieMallone.com`. You can get your own personalized *domain name*, if it's available. It's not free, but it's not expensive, either—about \$10 or \$15 a year. If you want to put your website address on a business card or a brochure for a small business, your own domain name is really the best choice. In Chapter 3, you'll learn how to buy one.

Note: The domain name is the first part of a web address, which identifies the web server that's storing and serving up your site. In the URL `www.ebay.com/help/index.html`, the domain name is `www.ebay.com`. You'll learn much more about domain names and URLs (short for Universal Resource Locator, also known as web addresses), and how they work, in Chapter 3.

- **Web design tools.** Creating web pages from scratch is a great way to learn a new skill, but it's far too slow and painful to create a complete website that way. To get to the next level, you need to step up to a professional web design tool. If you have a commercial program like Dreamweaver, you're in good hands. Even if you don't, many good free and shareware products can help you out. Chapter 4 explains your options and helps you get started.
- **Hyperlinks.** On its own, a single web page can do only so much. The real magic begins when you bind multiple pages together using links. Chapter 8 introduces the versatile hyperlink, which lets visitors move around your site.
- **Indispensable extras.** Once you master the basics of web pages and websites, there's far more ground to conquer. You can get your site listed in a search engine (Chapter 11), establish your own community forum (Chapter 12), and even sell stuff (Chapter 14). Still hungry for more? Why not create interactive pages with JavaScript code (Chapter 15), design a pop-up menu (Chapter 16), and add audio and video (Chapter 17)? All these features take you beyond basic web design and put you on the road to becoming a genuine web expert.

Creating Your First Page

Web pages are the basic unit of a website, and every website is a collection of one or more pages. The ideal web page contains enough information to fill the width of a browser window, but not so much that readers need to scroll from morning until lunchtime to get to the page's end. In other words, the ideal page strikes a balance: It avoids the lonely feeling caused by too much white space, as well as the stress induced by an avalanche of information.

The best way to get a handle on what a web page should hold is to look at your favorite websites. A news site like www.nytimes.com displays every news article on a separate page (and subdivides longer stories into several pages). At an e-commerce shop like www.amazon.com, every product has its own page. Similarly, a personal website like www.MyUndyingLoveForPigTrotters.com could be divided into separate web pages with titles like "About Me," "Vacation Photos," and "Top-Secret Recipes for Pig Parts."

For now, don't worry too much about how to divide your website into pages; you'll revisit that task in Chapter 8 when you start linking pages together. Instead, your first goal is to understand how a basic web page works and how to create one of your own. That's what you'll do in this chapter. On the way, you'll learn the essential details of the most important standard in website design: HTML.

Introducing HTML: The Language of the Web

Web pages are written in *HTML* (HyperText Markup Language). It doesn't matter whether your page contains a series of text-only blog entries, a dozen pictures of your pet lemur, or a heavily formatted screenplay—odds are that, if you're looking at it in a browser, it's an HTML page.

HTML plays a key role in web pages: It tells browsers how to display the contents of a web page, using special instructions (called *tags*) that instruct the browser when to start a paragraph, italicize a word, or display a picture. To create your own web pages, you need to learn to use this family of tags.

HTML is such an important standard that you'll spend a good portion of this book digging through its features, frills, and shortcomings. Every web page you build along the way will be a bona fide HTML document.

Note: The HTML standard doesn't have anything to do with the way a web browser *retrieves* a page on the Web. That task is left to HTTP (HyperText Transport Protocol), the low-level communication system that lets two computers exchange data over the Internet. If you were to apply the analogy of a phone conversation, the telephone line is HTTP, and the juicy tidbits of gossip you exchange with Aunt Martha are the HTML documents.

Cracking Open an HTML File

On the inside, an HTML page is actually nothing more than a plain-vanilla text file. That means that every web page consists entirely of letters, numbers, and a few special characters (like spaces, punctuation marks, and everything else you can spot on your keyboard). An HTML file is quite different from what you'd find if you cracked open a typical *binary file* on your computer. A binary file contains genuine computer language—a series of 0s and 1s. If another program is foolish enough to try to convert this binary information into text, you end up with gibberish.

To understand the difference between a text file and a binary file, take a look at Figure 2-1, which examines a traditional Word document (a *.doc* file) under the microscope. Compare that with what you see in Figure 2-2, which dissects an HTML document with the same content.

To examine an HTML document, all you need is an ordinary text editor. Windows computers come with one called Notepad. To open it in Windows Vista or Windows 7, click the Start button, type “Notepad” into the search box, and then click on the Notepad icon that appears. On a Mac, use its stock text editor,TextEdit, which you find at Applications→TextEdit.

In your text editor, you can open your HTML file, usually with the straightforward File→Open menu command. If you downloaded the companion content for this book (all of which you can find on the Missing CD page at www.missingmanuals.com/cds/caw3), you can try opening the *popsicles.htm* file shown in Figure 2-2.

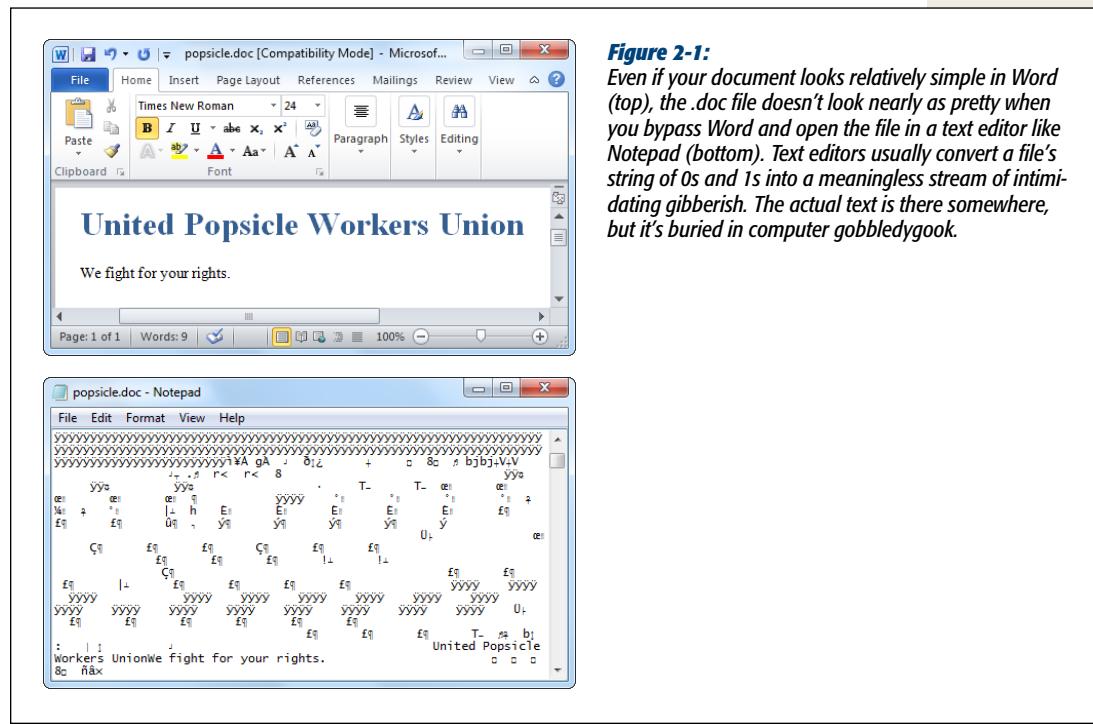


Figure 2-1:

Even if your document looks relatively simple in Word (top), the .doc file doesn't look nearly as pretty when you bypass Word and open the file in a text editor like Notepad (bottom). Text editors usually convert a file's string of 0s and 1s into a meaningless stream of intimidating gibberish. The actual text is there somewhere, but it's buried in computer gobbledegook.

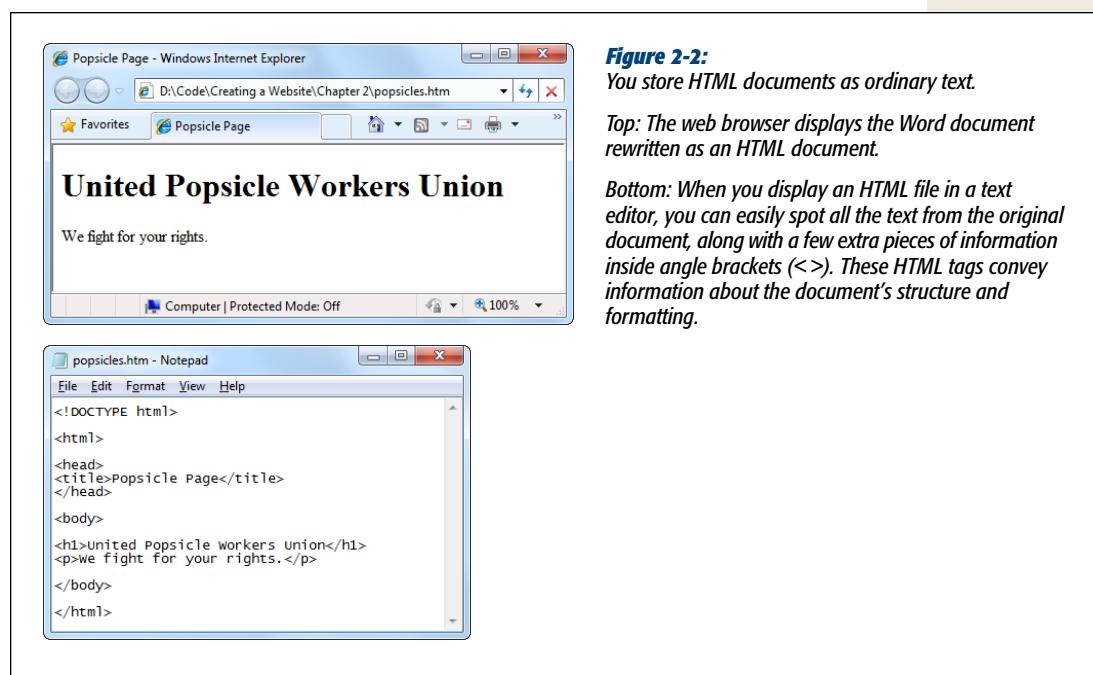


Figure 2-2:

You store HTML documents as ordinary text.

Top: The web browser displays the Word document rewritten as an HTML document.

Bottom: When you display an HTML file in a text editor, you can easily spot all the text from the original document, along with a few extra pieces of information inside angle brackets (<>). These HTML tags convey information about the document's structure and formatting.

Seeing the HTML of a Live Web Page

Most text editors don't let you open a web page directly from the Internet. To do that, they'd need to be able to send a request over the Internet to a web server, which is a job best left to a web browser. However, the browsers themselves *do* give you the chance to look at the raw HTML of a web page. Here's what to do:

1. Open your preferred browser.
2. Navigate to the web page you want to examine.
3. In your browser, right-click the page and then click View Source (in Internet Explorer) or View Page Source (in Firefox and Chrome).

Once you make your selection, a new window appears, showing you the raw HTML that creates the web page you see.

Tip: Firefox has a handy feature that lets you home in on part of the HTML in a complex page. Just select the text you're interested in on the web page itself, right-click the text, and then choose View Selection Source.

Most web pages are considerably more complex than the *popsicles.htm* example shown in Figure 2-2, so you need to wade through many more HTML tags. But once you acclimate yourself to the jumble of information, you'll have an extremely useful way to peer under the covers of any web page. In fact, professional web developers often use this trick to check out the work of their competitors.

Creating an HTML File

Here's one of the best-kept secrets of web page writing: You don't need a live website to start creating your own web pages. That's because you can easily build and test pages using only your own computer. In fact, you don't even need an Internet connection.

The basic approach is simple:

1. Fire up your favorite text editor.
2. Start writing HTML content.

Of course, this part is a little tricky because you haven't explored the HTML standard yet. Hang on—help is on the way in the sections below. For now, you can use the following very simple HTML snippet. Just type it in exactly as it appears, text, slashes, pointy brackets, and all:

```
<h1>United Popsicle Workers Union</h1>
<p>We fight for your rights.</p>
```

Technically, this two-line document is missing a few structural details that self-respecting web pages should have. However, every browser can read this HTML fragment and correctly interpret what you want: the two lines of formatted text shown in Figure 2-2.

3. When you finish your web page, save the document.

In Notepad, start by choosing File→Save. At the bottom of the Save As dialog box, in the Encoding list, choose UTF-8. (Your web page will work even if you don't take this step, but it keeps validators happy.)

In TextEdit, you need to first choose Format→Make Plain Text to make sure the program saves your page as an ordinary text file (and not a rich, binary document). Then, you can use File→Save to save the file. In the Plain Text Encoding box, choose Unicode (UTF-8).

When you name your file, use the extension *.htm* or *.html* (see the note below). For example, a typical HTML file name is LimeGreenPyjamas.html. If you use TextEdit, the program may ask if you really want to use the *.htm* or *.html* extension instead of *.txt*, the text file standard; click “Use *.htm*.” No such step is required in Notepad. However, in the “Save as type” box, you will need to choose All Files (*.*) instead of Text Documents (*.txt) if you want to see your HTML files in the file list.

Note: Technically speaking, you can use any file extension you want. However, using *.htm* or *.html* saves confusion (you immediately know the file is a web page) and helps avoid common problems. For example, using an *.htm* or *.html* file extension ensures that when you double-click the file name, your computer will know to open it in a web browser and not some other program). It's also important to use the *.htm* or *.html* extension if you plan to upload your files to a web server; prickly servers may refuse to hand out pages that have nonstandard file extensions.

4. To view your work, open the file in a web browser.

If you use the extension *.htm* or *.html*, opening a page is usually as easy as double-clicking the file name. If not, you may need to type the full file path into your web browser's address bar, as shown in Figure 2–3.

Remember, when you compose your HTML document in a text editor, you won't be able to see the formatted document. All you'll see is the plain text and the HTML formatting instructions.

Tip: If you change and save the file *after* you open it in your web browser, you can take a look at your recent changes by refreshing the page. In most browsers, that's as easy as right-clicking the page and choosing Refresh or Reload.

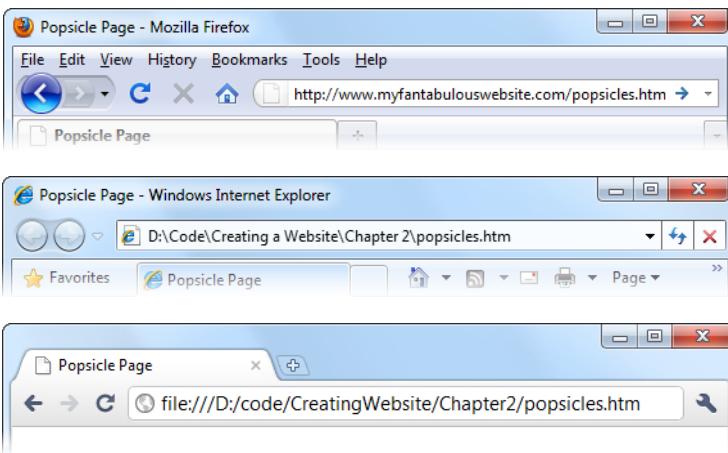


Figure 2-3:
A browser's address bar reveals where the current web page is really located. If you see "http://" in the address, it comes from a web server on the Internet (top). If you look at a web page on your own computer, you see just an ordinary file path (middle, showing a Windows file location in Internet Explorer), or you see a URL that starts with the prefix "file://" (bottom, showing a file location in Chrome). Local addresses depend on the browser and operating system you use.

TROUBLESHOOTING MOMENT

If Your First Web Page Doesn't Look Right...

...the trouble is probably in the way you saved the file.

For example, one common problem is having your document appear in the web browser without formatting and with all the HTML tags showing. In other words, your document looks the same in your browser as it does in your text editor. Any one of several oversights can cause this problem:

- **You used the wrong file extension.** When you open files directly from your computer (not from a remote website), your browser may attempt to identify the file type by its extension. If you give your web page the extension .txt, the browser may assume that it's a text file, and simply show the file's raw text content. To avoid this headache, use the .htm or .html extension.

- **You saved the document in a word processor.**

Word processors automatically convert special HTML characters, like angle brackets, into character entities (see page 130). For example, a word processor converts a tag like <h1> into actual text ("<h1>") rather than recognizing it as a raw HTML formatting instruction. To avoid this, write your HTML in a text editor.

- **You didn't save it as a text file.** Some text editors let you save your documents in different formats.

Notepad doesn't, butTextEdit does, and if you inadvertently save your web page as a rich file, a browser won't treat it as a web page. To avoid this problem inTextEdit, make sure you switch on the setting Format→Make Plain Text. In other programs, check for options in the File→Save As window.

HTML, XHTML, and HTML5

Since its introduction, HTML has evolved in new and sometimes unexpected directions. Recently, the situation has become a bit messy.

HTML is the original language of the Web. The last completed version of HTML (version 4.01) was set in stone at the end of 1999. For the last decade, HTML has been gradually overtaken by a closely related, modernized version of the language called XHTML (*Extensible HyperText Markup Language*).

Note: Because XHTML is a variant of HTML and because HTML was the original standard of the Web, it's common to refer to all web pages as being written in HTML (as this book does). When used in this way, the term "HTML" actually means "any version of HTML or XHTML."

Here's the messy part: The version of XHTML that's in use today is XHTML 1.0 or (less commonly) XHTML 1.1. Developers were supposed to replace both these versions of XHTML with an even newer standard called XHTML 2.

Unfortunately, the XHTML 2 developers lost the plot. They created a standard that was philosophically pure but practically a nightmare. Web developers and the big browser vendors were scared by the strictness of XHTML 2 and the fact that it wasn't backward-compatible with existing web pages. In other words, there was no easy way to turn a respectable XHTML 1.0 page into a snazzy new XHTML 2 one. Furthermore, the XHTML 2 standard lacked the fancy new features that many web developers, particularly those building super-complex web applications, *really* wanted.

The final blow for XHTML 2 came when a competing group began presenting early versions of a standard called HTML5. Although HTML5 is itself new, different, and not yet fully supported by any browser in the world, it has generated plenty of excitement. In fact, the W3C (the standards organization officially in charge of HTML and XHTML) recently abandoned work on the doomed XHTML 2 standard and began formalizing HTML5, making it official: HTML5 is the future of the Web.

Note: There's no space between the "HTML" and the "5" in HTML5. This is just a quirk in the way the standard is named.

Confused? Don't worry—the designers of HTML5 were smart. They made it work with every feature that works today in XHTML. That means you can write HTML5 documents right now (as you will in this book), without using any of the new-fangled features that browsers don't yet support, and everything will work out just fine. Best of all, you'll be ready for the not-so-distant future, when new browsers colonize the earth and you can finally start using some of the fancy HTML5-only features.

Note: HTML5 supports everything that's possible in XHTML—with no changes required. However, the syntax of XHTML is stricter than that of HTML5, and there are several practices that HTML5 supports but XHTML does not. Web developers are divided over whether these practices are horribly bad style or just convenient shortcuts. To encourage you to stick to good habits, this book always uses the stricter XHTML-style syntax.

HTML5 PREVIEW

A Bit More About HTML5

Wrapping your head around HTML5 can be a bit of a challenge. Part of the problem is that the word “HTML5” actually includes the latest version of the HTML language and a pile of emerging standards that are still under development. And just to make life more interesting, some of these standards work with many of today’s most popular browsers, and others aren’t yet implemented even on the latest versions of HTML5.

Interestingly, HTML5 offers only a few additions to the standard family of tags you’ll learn to use in this chapter. Most of these tags give web browsers, search engines, and other automated tools a bit more structural information about your web pages. For example, you can use specialized tags to tell a browser what section of your page holds its navigation links.

More glamorously, HTML5 also includes a new way to add video to your web page without requiring a plug-in, as

most browsers do today. (Chapter 17 has more about this feature.) But the chief goal of HTML5, and its related standards, is to provide a set of powerful tools for developing web *applications*, web pages you interact with. Examples include a canvas for drawing two-dimensional pictures, a geolocation feature that pinpoints your visitor’s geographic location, and a technique that lets interactive web pages work offline. All these features are planted firmly on the bleeding edge of web design, and none of them works in versions of Internet Explorer before IE 9.

If you’re curious about the future, keep looking for the “HTML5 Preview” boxes in this book. You can also check out Appendix A for a brief outline of some of the new elements in the HTML5 language. To get the full scoop on the eventual future of web development, check out *HTML5: The Missing Manual*.

HTML Tags

Now that you know how to peer into existing HTML files and how to create your own, the next step is to understand what goes *inside* the average HTML file. It all revolves around a single concept—*tags*.

HTML tags are formatting instructions that tell a browser how to transform ordinary text into something visually appealing. If you were to take all the tags out of an HTML document, the resulting page would consist of nothing more than plain, unformatted text.

What’s in a Tag

You can recognize a tag by looking for angle brackets, two special characters that look like this: < >. To create a tag, you type HTML code between the brackets. This code is for the browser’s eyes only; web visitors never see it (unless they use the View→Source trick to peek at the HTML). Essentially, the code is an instruction that conveys information to the browser about how to format the text that follows.

For example, one simple tag is the tag, which stands for “bold” (tag names are always lowercase). When a browser encounters this tag, it switches on boldface formatting, which affects all the text that follows the tag. Here’s an example:

This text isn't bold. **This text is bold.**

On its own, the **** tag isn't quite good enough; it's known as a *start tag*, which means it switches on some effect (in this case, bold lettering). Most start tags are paired with a matching *end tag* that switches *off* the effect.

You can easily recognize an end tag. They look the same as start tags, except that they begin with a forward slash. They look like this **</>** instead of like this **<>**. So the end tag for bold formatting is ****. Here's an example:

This isn't bold. **Pay attention!** Now we're back to normal.

Which a browser displays as:

This isn't bold. **Pay attention!** Now we're back to normal.

This example reflects another important principle of browsers: They always process tags in order, based on where the tags show up in your text. To get the bold formatting in the right place, you need to make sure you position the **** and **** tags appropriately.

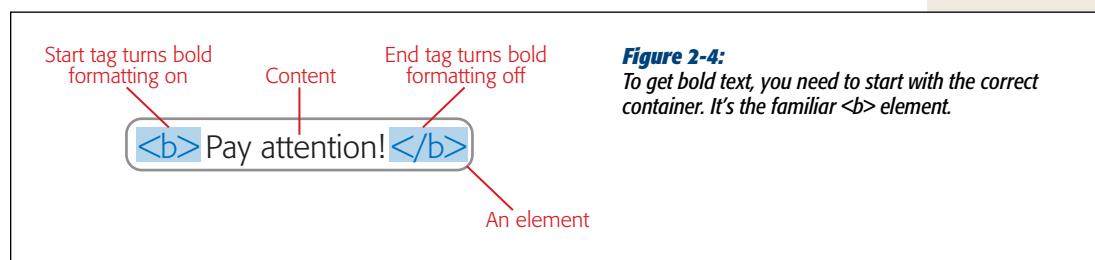
As you can see, the browser has a fairly simple job. It scans an HTML document, looking for tags and switching on and off various formatting settings. It takes everything else (everything that isn't a tag) and displays it in the browser window.

Note: Adding tags to plain-vanilla text is known as *marking up* a document, and the tags themselves are known as HTML *markup*. When you look at raw HTML, you may be interested in looking at the content (the text nestled between the tags) or the markup (the tags themselves).

Understanding Elements

As you've seen, tags come in pairs. When you use a start tag (like **** for bold), you have to include an end tag (like ****). This combination of start and end tags and the text in between them makes up an HTML *element*.

Here's the basic idea: Elements are containers (see Figure 2-4). You place some content (like text) inside that container. For example, when you use the **** and **** tags, you create a container that applies bold formatting to the text inside the container. As you create your web pages, you'll use different containers to wrap different pieces of text. If you think about elements this way, you'll never forget to include an end tag.



Note: When someone refers to the ` element`, they mean the whole shebang—start tag, end tag, and the content in between. When someone refers to a ` tag`, they simply means the instruction that triggers the effect.

Of course, life wouldn't be much fun (and computer books wouldn't be nearly as thick) without exceptions. When you get right down to it, there are really *two* types of elements:

- **Container elements** are, by far, the most common type. They apply formatting to the content nestled between the start and end tags.
- **Standalone elements** don't turn formatting on or off. Instead, they *insert* something, like an image, into a page. One example is the `
` element, which inserts a line break in a page. Standalone elements don't come in pairs, as container elements do, and you may hear them referred to as *empty* elements because you can't put any text inside them.

In this book, all standalone elements include a slash character before the closing `>`, sort of like an opening and closing tag all rolled into one. So you'll see a line break written as `
` instead of as `
`. This form, called the *empty element syntax*, is handy because it clearly separates the container elements from the standalone elements. That way, you'll never get confused.

Note: In the not-so-distant past, web developers were forced to use the empty element syntax, because it was an official part of the XHTML language. However, HTML5 makes this convention optional and allows standalone elements to use the same syntax as start tags (which means you can use either `
` or `
` to insert a line break, for instance).

Figure 2-5 puts the two types of elements in perspective.

This container element holds a word

Just `` do `` it. `
` Just say no.

This stand-alone element inserts a line break

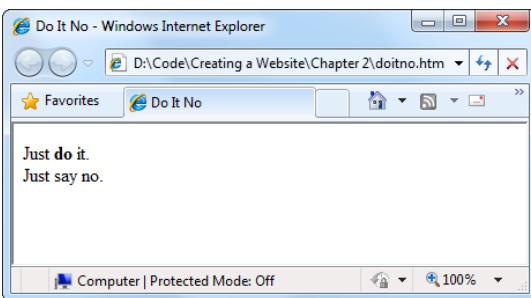


Figure 2-5:

Top: This snippet of HTML shows both a container element and a standalone element.

Bottom: The browser shows the resulting web page.

Nesting Elements

In the previous example, you applied a simple `` element to get bold formatting. You put the text you wanted to make bold between the `` and `` tags. However, text isn't the only thing you can put between a start and end tag. You can also nest one element *inside* another. In fact, nesting elements is one of the basic building-block techniques of web pages. Nesting lets you apply more detailed formatting instructions to text (for example, it lets you create bold, italicized text) by combining all the elements you need in the same set of instructions. You can also nest elements to create more complicated page components, like bulleted lists.

To see nesting in action, you need another element to work with. For the next example, consider both the familiar `` element and the `<i>` element, which italicizes text.

The question is, what happens if you want to make a piece of text bold *and* italicized? HTML doesn't include a single element for this purpose, so you need to combine the two. Here's an example:

This `<i>word</i>` has italic and bold formatting.

When a browser chews through this scrap of HTML, it produces text that looks like this:

This *word* has italic and bold formatting.

Incidentally, it doesn't matter if you reverse the order of the `<i>` and `` tags. The following HTML produces exactly the same result.

This `<i>word</i>` has italic and bold formatting.

However, you should always make sure that you close tags in the *reverse* order from which you opened them. In other words, if you apply italic formatting and then bold formatting, you should switch off bold formatting first, and then italic formatting. Here's an example that breaks this rule:

This `<i>word</i>` has italic and bold formatting.

Browsers can usually sort this out and make a good guess about what you really want, but it's still a dangerous habit as you write more complex HTML.

As you'll see in later chapters, HTML gives you many more ways to nest elements. For example, you can nest one element inside another, and then nest another element inside that one, and so on, indefinitely.

Note: If you're a graphic-design type, you're probably itching to get your hands on more powerful formatting elements to change alignment, spacing, and fonts. Unfortunately, in the web world you can't always control everything you want. Chapter 5 has the lowdown, and Chapter 6 introduces the best solution, style sheets.

FREQUENTLY ASKED QUESTION

Telling a Browser to Ignore a Tag

What if I really do want the text “``” to appear on my web page?

The tag system works great until you actually want to use an angle bracket (`<` or `>`) in your text. Then you’re in a tricky position.

For example, imagine you want to write the following bit of text as proof of your remarkable insight:

The expression `5 < 2` is clearly false,
because `5` is bigger than `2`.

When a browser reaches the less-than (`<`) symbol, it becomes utterly bewildered. Its first instinct is to assume you’re starting a tag, and the text “`2` is clearly false...” is part of a long tag name. Obviously, this isn’t what you intended, and it’s certain to confuse the browser.

To solve this problem, you need to replace angle brackets with the corresponding HTML *character entity*. Character entities always begin with an ampersand (&) and end with a semicolon (;). The character entity for the less-than symbol

is `<` because the `l` stands for “less than.” Similarly, `>` is the character entity for the greater-than symbol.

Here’s the corrected example:

The expression `5 < 2` is clearly false,
because `5` is bigger than `2`.

In your text editor, this doesn’t look like what you want. When a browser displays this document, however, it automatically changes the `<` into a `<` character, without confusing it with a tag. You’ll learn more about character entities on page 130.

Incidentally, character replacement is one of the reasons you can get into trouble if you attempt to write your HTML documents in a word processor. When you save your word processor document as a text file, the word processor converts all the special characters into the corresponding character entities. As a result, you see ordinary text in your browser, not the formatting you expect.

The HTML Document

So far, you’ve been considering HTML snippets—portions of a complete HTML document. In this section, you’ll learn how to put it all together and create your first genuine web page.

The Document Type Definition

In the early days of the Internet, web browsers were riddled with quirks. When people designed web pages, they had to take these quirks into account. For example, browsers might calculate the margins around floating boxes of text in subtly different ways, causing pages to look right in one browser, but appear odd in another.

Years later, the rules of HTML (and CSS, the style sheet standard you’ll consider in Chapter 6) were standardized in more detail. Using these new rules, every browser could display the same page in exactly the same way. But this change caused a serious headache for longstanding browsers that had lived through the dark ages of HTML, like Internet Explorer. These browsers had to somehow support the new

standards, while still being able to properly display existing web pages—including those that relied on old quirks.

The web community settled on a simple solution. When designing a new, modern web page, you indicate this fact by adding a code called a *document type definition* (DTD) or *doctype*, which goes at the very beginning of your HTML document (Figure 2-6).

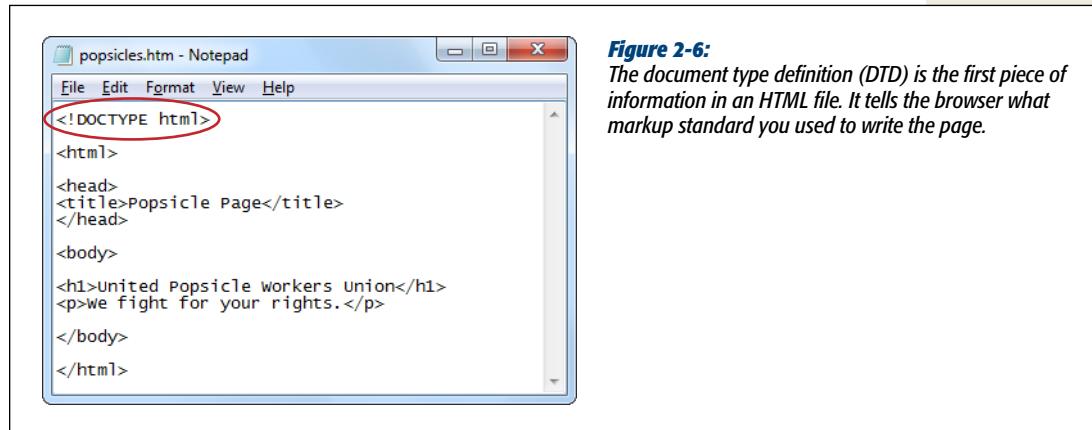


Figure 2-6:

The document type definition (DTD) is the first piece of information in an HTML file. It tells the browser what markup standard you used to write the page.

When a browser encounters a page with a doctype, it switches into *standards mode*. It then renders the page in the most consistent, standardized way possible. The end result is that the page looks virtually identical in every modern browser.

But when a browser encounters an HTML document that doesn't have a doctype, all bets are off. Some, like Internet Explorer, switch into the dreaded *quirks mode*. When in quirks mode, Internet Explore attempts to behave the same way earlier versions did 10 years ago, quirks and all. This ensures that really old web pages look the way they did back when they were first created, even if they rely on ancient browser bugs that have long since been fixed. Unfortunately, different browsers have different idiosyncrasies. So when you view a page with no doctype in multiple browsers, you're likely to get varying text sizes, inconsistent margins and borders, and improperly positioned content. For that reason, web pages without doctypes are bad news, and you should avoid creating them.

Web designers can use different doctypes to indicate the markup standard they're using (for example, XHTML, HTML5, or truly old HTML 4.01). However, browsers have a dirty doctype secret: They don't actually care what your doctype says. They simply care that you *have* a doctype. That's because almost all doctypes trigger standards mode. So you could give your web page an XHTML 1.0 doctype but put HTML5 content inside, and your browser wouldn't mind at all.

Today, most web developers use the HTML5 doctype, which looks like this:

```
<!DOCTYPE html>
```

You'll notice that this HTML5 doctype doesn't indicate the version number "5." Instead, it just indicates that the language is HTML. This isn't a typo. Instead, it reflects the philosophy of HTML—to support documents old and new. This also means that as new features are added to HTML, they will be automatically available in your existing web pages. (This is actually the way that browsers already work. HTML5 just makes it official.)

For comparison, here's the much wordier doctype for XHTML 1.0, which you're likely to spot in slightly older web pages:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Even seasoned web developers had to copy the XHTML 1.0 doctype from an existing web page to avoid typing it in wrong.

In this book, all of the examples use the HTML5 doctype. Even though no browsers support all of the new stuff in HTML5 (and some browser versions don't support any of it), all browsers accept the HTML5 doctype and switch into standards mode when they see it. That means that any web page with the HTML5 doctype gets the same, consistent rendering it would with the traditional XHTML 1.0 doctype.

Note: The advantage of using the HTML5 doctype is that it prepares your pages for the future. Just because you added the doctype, however, don't assume you can use new HTML5-specific features. In fact, you should do your best to avoid most of these features for the time being, because they aren't widely supported.

The Basic Skeleton

Now that you've chosen your doctype, you're ready to fill in the rest of your web page.

To create a true HTML document, you start with three container elements: `<html>`, `<head>`, and `<body>`. These three elements work together to describe the basic structure of your page:

`<html>`

This element wraps everything (other than the doctype) in your web page.

`<head>`

This element designates the *header* portion of your document. The header can include some optional information about your web page, including the required title (which your browser displays in its title bar), optional search keywords, and an optional style sheet (which you'll learn about in Chapter 6).

`<body>`

This element holds the meat of your web page, including the actual content you want to display to the world.

There's only one right way to combine these three elements. Here's their correct arrangement, with the HTML5 doctype at the beginning of the page:

```
<!DOCTYPE html>

<html>

<head>
...
</head>

<body>
...
</body>

</html>
```

Every web page uses this basic framework. The ellipsis (...) shows where you insert additional information. The spaces between the lines aren't required—they're just there to help you see the element structure more easily.

Once you have the HTML skeleton in place, you need to add two more container elements to the mix. Every web page requires a `<title>` element, which goes in the header section. Next, you need to create a container for text in the `<body>` section of the page. One all-purpose text container is the `<p>` element, which represents a paragraph.

Here's a closer look at the elements you need to add:

`<title>`

This element sets the title for your web page. The title plays several roles. First, web browsers display the title at the top of the browser window. Second, when a web visitor bookmarks your page, the browser uses the title to label the bookmark in your Bookmark (or Favorites) menu. Third, when your page turns up in a web search, the search engine usually displays this title as the first line in the search results, followed by a snippet of content from the page.

`<p>`

This indicates a paragraph. Web browsers don't indent paragraphs, but they do add a little space between consecutive paragraphs to keep them separated.

Here's the web page with these two new ingredients:

```
<!DOCTYPE html>

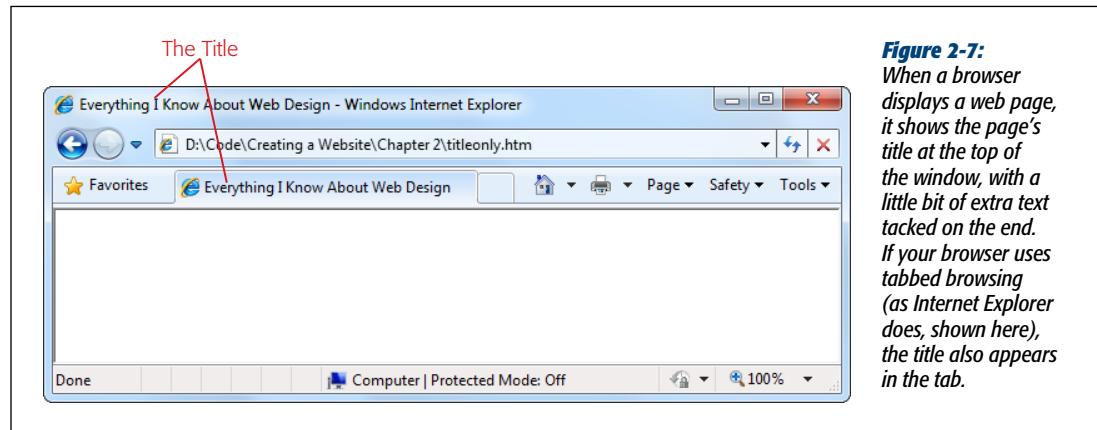
<html>

<head>
<title>Everything I Know About Web Design</title>
</head>

<body>
<p></p>
</body>

</html>
```

If you open this document in a web browser, you'll find that the page is blank, but the title appears (as shown in Figure 2-7).



As it stands right now, this HTML document is a good template for future pages. The basic structure is in place; you simply need to change the title and add some text.

Adding Content

You're finally ready to transform this HTML skeleton into a real document by adding content. This detail is the most important, and it's the task you'll work on through most of this book.

For example, suppose you're writing a simple résumé page. Here's a very basic first go at it:

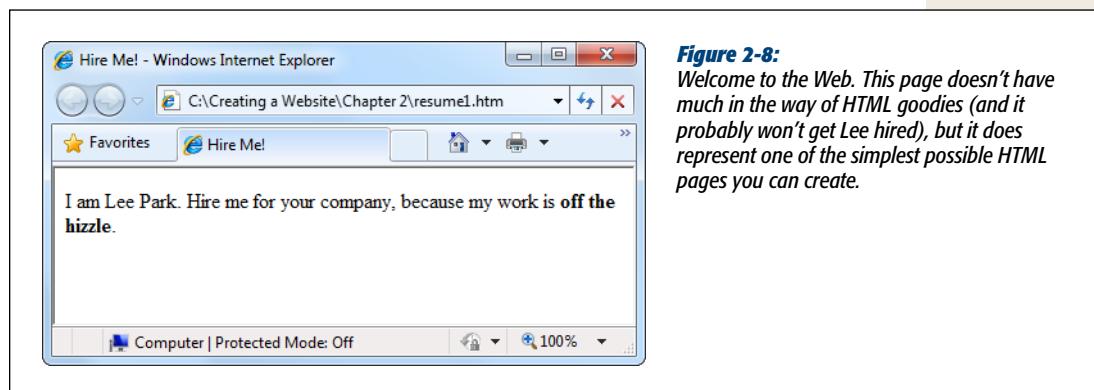
```
<!DOCTYPE html>

<html>

<head>
<title>Hire Me!</title>
</head>
<body>
<p>I am Lee Park. Hire me for your company, because my work is <b>off
the hizzle</b>.</p>
</body>

</html>
```

This example highlights (**in bold**) the modifications made to the basic HTML structure—a changed title and a single line of text. It uses a single **** element inside the paragraph, just to dress up the page a little. Before you go any further, you may want to try creating this sample file in your own text editor (using the process you learned on page 22) and opening it in your favorite web browser (see Figure 2-8).

**Figure 2-8:**

Welcome to the Web. This page doesn't have much in the way of HTML goodies (and it probably won't get Lee hired), but it does represent one of the simplest possible HTML pages you can create.

Using the HTML tricks described in the following sections you can build on this example and give Lee Park a better résumé.

Tip: Even if you have high-powered HTML editing software like Dreamweaver, don't use it yet. When you start learning HTML, do it by hand so you understand every part of your web page. Later on, when you've mastered the basics and are ready to create more sophisticated web pages, you'll probably want to switch to other tools, as discussed in Chapter 4.

Structuring Text

As you start to create more detailed web pages, you'll quickly discover that building a page isn't as straightforward as, say, creating a page in Microsoft Word. For example, you may decide to enhance the résumé page by creating a list of skills. Here's a reasonable first try:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hire Me!</title>
  </head>
  <body>
    <p>I am Lee Park. Hire me for your company, because my work is <b>off the hizzle</b>.


My skills include:



- Fast typing (nearly 12 words/minute).
- Extraordinary pencil sharpening.
- Inventive excuse making.
- Negotiating with officers of the peace.</p>

  </body>
</html>
```

The trouble appears when you open this seemingly innocent document in your web browser (Figure 2-9).

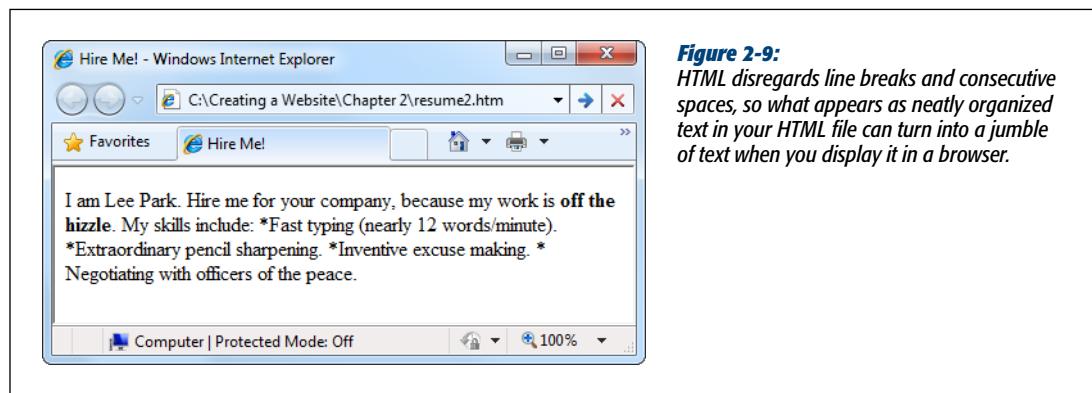


Figure 2-9:

HTML disregards line breaks and consecutive spaces, so what appears as neatly organized text in your HTML file can turn into a jumble of text when you display it in a browser.

The problem is that HTML ignores extra white space. That includes tabs, line breaks, and extra spaces (anything more than one consecutive space). The first time this happens, you'll probably stare at your screen dumbfounded and wonder why web browsers are designed this way. But it actually makes sense when you consider that HTML needs to work as a *universal standard*.

Say you customized your hypothetical web page with the perfect spacing, indenting, and line width for *your* computer monitor. The hitch is, this page may not look as good on someone else's monitor. For example, some of the text may scroll off the right side of the page, making it difficult to read. And different monitors are only part of the problem. Today's web pages need to work on different types of *devices*. Lee Park's future boss might view Park's résumé on anything from the latest wide-screen laptop to a tablet computer or smartphone.

To deal with this range of display options, HTML uses elements to define the *structure* of your document. Instead of telling the browser, "Here's where you go to the next line and here's where you add four extra spaces," HTML tells the browser, "Here are two paragraphs and a bulleted list." It's up to the browser to display the page, using the instructions you include in your HTML.

To correct the résumé example, you need to use more paragraph elements and two new container elements:

Indicates the start of an unordered list. A list is the perfect way to detail Lee's skills.

Indicates an individual item in a bulleted list. Your browser indents each list item and, for sentences that go beyond a single line, properly indents subsequent lines so they align under the first. In addition, it precedes each item with a bullet (•). You can use the element only inside a list element like . In other words, every list item needs to be a part of a list.

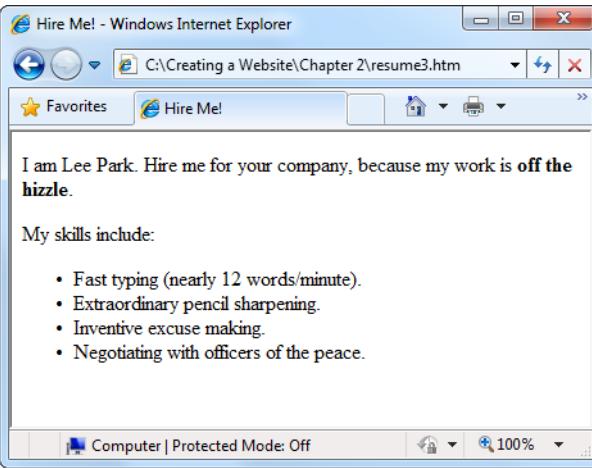
Here's the corrected web page (shown in Figure 2-10), with the structural elements highlighted in bold:

```
<!DOCTYPE html>

<html>

<head>
<title>Hire Me!</title>
</head>
<body>
<p>I am Lee Park. Hire me for your company, because my work is <b>off the hizzle</b>.</p>
<p>My skills include:</p>
<ul>
<li>Fast typing (nearly 12 words/minute).</li>
<li>Extraordinary pencil sharpening.</li>
<li>Inventive excuse making.</li>
<li>Negotiating with officers of the peace.</li>
</ul>
</body>

</html>
```



The screenshot shows a Windows Internet Explorer window titled "Hire Me! - Windows Internet Explorer". The address bar displays the file path "C:\Creating a Website\Chapter 2\resume3.htm". The main content area of the browser shows the following HTML code as rendered by the browser:

```
I am Lee Park. Hire me for your company, because my work is off the hizzle.

My skills include:



- Fast typing (nearly 12 words/minute).
- Extraordinary pencil sharpening.
- Inventive excuse making.
- Negotiating with officers of the peace.

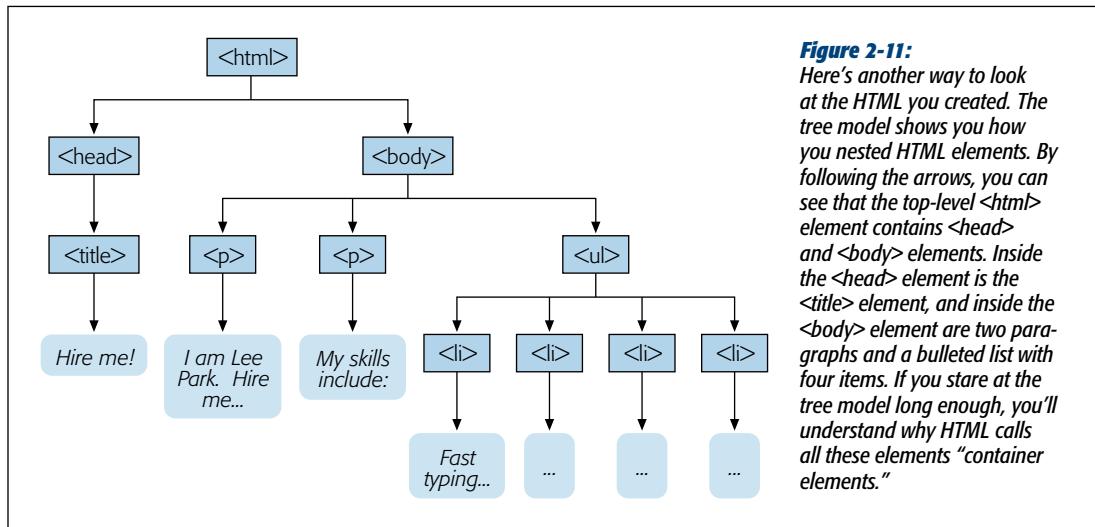
```

The browser interface includes standard navigation buttons (Back, Forward, Stop, Home) and status bars at the bottom indicating "Computer | Protected Mode: Off" and "100%".

Figure 2-10:
With the right elements (as shown in the code on this page), the browser understands the structure of your HTML document, and knows how to display it.

You can turn a browser's habit of ignoring line breaks to your advantage. To help make your HTML documents more readable, add line breaks and spaces wherever you want. Web experts often use indentation to make the structure of nested elements easier to understand. In the résumé example, you can see this trick in action. Notice how the list items (the lines starting with the `` element) are indented. This has no effect on the browser, but it makes it easier for you to see the structure of the HTML document and to gauge how a browser will render it.

Figure 2-11 analyzes the HTML document using a *tree model*. The tree model is a handy way to get familiar with the anatomy of a web page, because it shows the page's overall structure at a glance. However, as your web pages get more complicated, they'll probably become too complex for a tree model diagram to be useful.



If you're a masochist, you don't need to use any spaces. The previous example is exactly equivalent to the following much-less-readable HTML document:

```
<!DOCTYPE html><html><head><title>Hire Me!</title></head><body><p>I am  
Lee Park. Hire me for your company, because my work is <b>off the hizzle  
</b>.</p><p>My skills include:</p><ul><li>Fast typing (nearly 12  
words/minute).</li><li> Extraordinary pencil sharpening.</li><li>Inventive  
excuse making.</li><li> Negotiating with officers of the peace.</li></ul>
```

Of course, it's nearly impossible for a human to write HTML like this without making a mistake.

GEM IN THE ROUGH

Have Something to Hide?

When you're working with a complex web page, you may want to temporarily remove an element or a section of content. This is a handy trick when you have a page that doesn't quite work right and you want to find out which elements are causing the problem. One way to do this is with the good ol' fashioned cut-and-paste feature in your text editor. Cut the section you think may be troublesome, save the file, and then load it up in your browser. If the section is innocent, paste it back in place, and then re-save the file. Repeat this process until you find the culprit.

However, HTML gives you a simpler solution—*comments*. With comments, you can leave the entire page intact. When you “comment out” a section of the page, HTML ignores it.

You create an HTML comment using the character sequence <!-- to mark the start of the comment, and the character sequence --> to mark the end. Your browser will ignore everything in between these two markers, whether it's content or tags. The comment markers can appear on the same line, or you can use them to hide an entire

section of your HTML document. However, don't try to nest one comment inside another, as that won't work.

Here's an example that hides two list items. When you open this document in your web browser, the list will show only the last two items (“Inventive excuse making” and “Negotiating with officers of the peace”).

```
<ul>
<!-- <li>Fast typing (nearly 12 words/
minute).</li>
-->
<li>Extraordinary pencil sharpening.</li>
<li>Inventive excuse making.</li>
<li>Negotiating with officers of the
peace.</li>
</ul>
```

When you want to return the list to its original glory, just remove the comment markers.

Where Are All the Pictures?

Whether it's a stock chart, a logo for your underground garage band, or a doctored photo of your favorite celebrity, the Web would be a pretty drab place without pictures. So far, you've seen how to put text into an HTML document, but what happens when you need an image?

Although it may seem surprising, you can't store a picture inside an HTML file. There are plenty of good reasons why you wouldn't want to anyway—your web page files would become really large, it would be hard to modify your pictures or do other things with them, and you'd have a fiendish time editing your pages in a text editor because the image data would make a mess. The solution is to store your pictures as separate files, and then *link* them to your HTML document. This way, your browser pulls up the pictures and positions them exactly where you want them on your web page.

The linking tool that performs this trick is the `` element (short for “image”). The `` element points to an image file, which the browser retrieves and inserts into the page. You can put the image file in the same folder as your web page (which is the easiest option) or you can put it on a completely different website.

Although you'll learn everything you ever wanted to know about web graphics in Chapter 7, it's worth considering a simple example right now. To try this out, you need a web-ready image handy. (The most commonly supported image file types are JPEG, GIF, and PNG.) If you downloaded this book's companion content (from the Missing CD page at www.missingmanuals.com/cds/caw3), you can use the sample picture *leepark.jpg*. Assuming you put this file in the same folder as your web page file, you can display the image with the following image element:

```

```

Like the *
* element discussed earlier, the ** element is a standalone element with no content. For that reason, it makes sense to use the empty element syntax, and add a forward slash before the closing angle bracket.

However, there's an obvious difference between the *
* element and the ** element. Although ** is a standalone element, it isn't self-sufficient. In order for the element to mean anything, you need to supply two more pieces of information: the name of the image file, and some alternate text, which is used in cases where your browser can't download or display the picture (see page 184). To incorporate this extra information into the image element, HTML uses *attributes*. Attributes are extra pieces of information that appear *after* an element name, but before the closing *>* character.

The ** example includes two attributes, separated by a space. Each attribute has two parts—a name (which tells the browser what the attribute does) and a value (a variable piece of information you supply). The name of the first ** attribute is *src*, which is shorthand for source; it tells the browser where to get the image you want. In this example, the value of the *src* attribute is *leepark.jpg*, which is the name of the file with Lee Park's headshot.

The name of the second ** attribute is *alt*, which is shorthand for “alternate text”; it tells a browser that you want it to show text if it can't display the image. Its value is the text you want to display, which is “Lee Park Portrait” in this case.

Once you've unraveled the image element, you're ready to use it in an HTML document. Just place it inside an existing paragraph, wherever it makes sense.

```
<!DOCTYPE html>

<html>

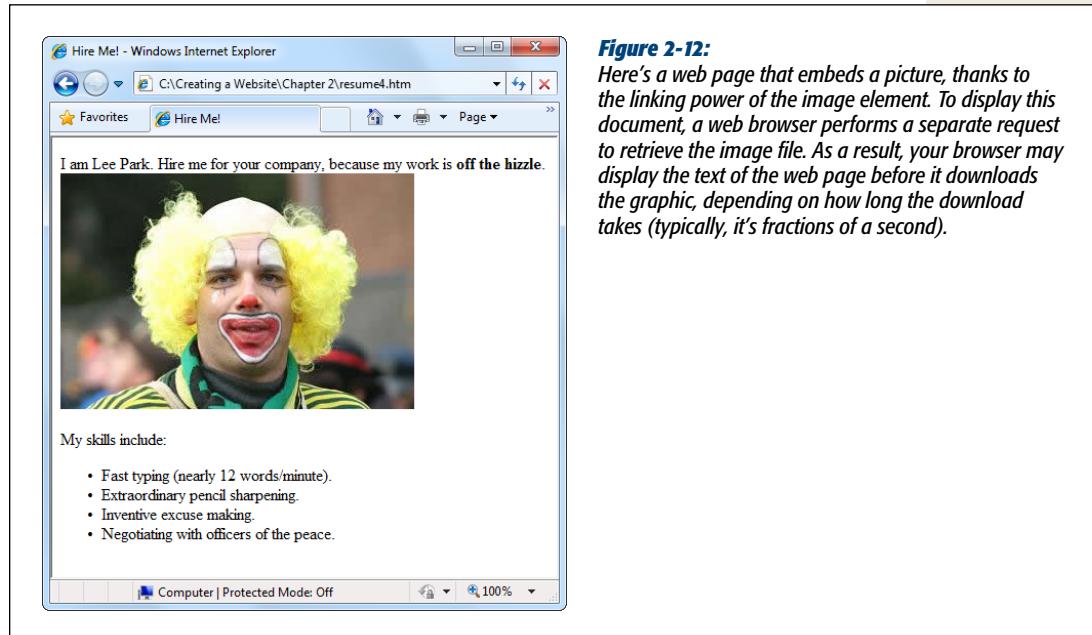
  <head>
    <title>Hire Me!</title>
  </head>

  <body>
    <p>I am Lee Park. Hire me for your company, because my work is <b>off the hizzle</b>.
    
    </p>
    <p>My skills include:</p>
    <ul>
      <li>Fast typing (nearly 12 words/minute).</li>
    </ul>
  </body>
</html>
```

```
<li>Extraordinary pencil sharpening.</li>
<li>Inventive excuse making.</li>
<li>Negotiating with officers of the peace.</li>
</ul>
</body>

</html>
```

Figure 2-12 shows exactly where the picture ends up.



Note: You'll learn many more tricks for web graphics, including how to change their size and wrap text around them, in Chapter 7.

The 10 Most Important Elements (and a Few More)

You've now reached the point where you can create a basic HTML document, and you already have several elements under your belt. You know the fundamentals—all that's left is to expand your knowledge by learning how to use more elements.

HTML has a relatively small set of elements. You'll most likely use fewer than 25 elements on a regular basis. This is a key part of its success, because it's made HTML into a simple shared language that anyone can understand and everyone can agree on.

Note: You can't define your own elements and use them in an HTML document because web browsers won't know how to interpret them.

Some elements, like the `<p>` element that formats a paragraph, are important for setting out the overall structure of a page. These are called *block-level elements*. You can place block-level elements directly inside the `<body>` section of your web page or, sometimes, inside another block-level element. Table 2-1 provides a quick overview of some of the most fundamental block-level elements, several of which you've already seen. It also points out which of these are container elements and which are standalone elements. (As you learned on page 28, container elements require start and end tags, but standalone elements get by with just a single tag.)

Table 2-1. Basic block-level element.

Element	Name	Type of Element	Description
<code><p></code>	Paragraph	Container	As your high school English teacher probably told you, the paragraph is the basic unit for organizing text. When you use more than one paragraph element in a row, a browser inserts a certain amount of space between the two paragraphs—just a bit more than a full blank line. Complete details appear in Chapter 5.
<code><h1>,<h2>, <h3>,<h4>, <h5>,<h6></code>	Heading	Container	Heading elements are a good way to add structure to your web page and make titles stand out. They display text in large, boldfaced letters. The lower the number, the larger the text, so <code><h1></code> produces the largest heading. By the time you get to <code><h5></code> , the heading has dwindled to the same size as normal-size text, and <code><h6></code> , although bold, is actually smaller than normal text.
<code><hr></code>	Horizontal Line	Standalone	A horizontal line can help you separate one section of your web page from another. The line automatically matches the width of the browser window. (Or, if you put the line inside another element, like a cell in a table, it takes on the width of its container.)
<code>,</code>	Unordered List, List Item	Container	These elements let you build basic bulleted lists. The browser automatically puts individual list items on separate lines and indents each one. For a quick change of pace, you can substitute <code></code> with <code></code> to get an automatically numbered list instead of a bulleted list (<code>ol</code> stands for “ordered list”).

Other elements are designed to deal with smaller structural details—for example, snippets of bold or italicized text, line breaks, links that lead to other web pages, and images. These elements are called *inline elements*. You can't put inline elements directly inside the `<body>` section. Instead, they have to be nested *inside* a block-level element. Table 2-2 lists the most useful inline elements.

Table 2-2. Basic inline elements.

Element	Name	Type	Description
<code>, <i></code>	Bold and Italic	Container	These two elements apply character styling—either bold or italic text. (Technically, <code><i></code> means “text in an alternate voice” and <code></code> means “stylistically offset text,” and there are tricks to change the formatting they apply, as you’ll see in Chapter 6. But in the real world, almost all web developers expect that <code><i></code> means italics and <code></code> means bold.)
<code>
</code>	Line Break	Standalone	Sometimes, all you want is text separated by simple line breaks, not separate paragraphs. This keeps subsequent lines of text closer together than when you use a paragraph. You’ll learn more about text layout in Chapter 5.
<code></code>	Image	Standalone	To display an image inside a web page, use this element. Make sure you specify the <code>src</code> attribute to indicate the file name of the picture you want the browser to show.
<code><a></code>	Anchor	Container	The anchor element is the starting point for creating hyperlinks that let website visitors jump from one page to another. You’ll learn about this indispensable element in Chapter 8.

To make the sample résumé look more respectable, use a few tricks from Tables 2-1 and 2-2.

```
<!DOCTYPE html>

<html>

<head>
  <title>Hire Me!</title>
</head>

<body>
  <h1>Hire Me!</h1>
  <p>I am Lee Park. Hire me for your company, because my work is <b>off the hizzle</b>. As proof of my staggering computer skills and monumental work ethic, please enjoy this electronic resume.</p>

  <h2>Indispensable Skills</h2>
  <p>My skills include:</p>
```

```

<ul>
    <li>Fast typing (nearly 12 words/minute).</li>
    <li>Extraordinary pencil sharpening.</li>
    <li>Inventive excuse making.</li>
    <li>Negotiating with officers of the peace.</li>
</ul>
<p>And I also know XHTML!</p>

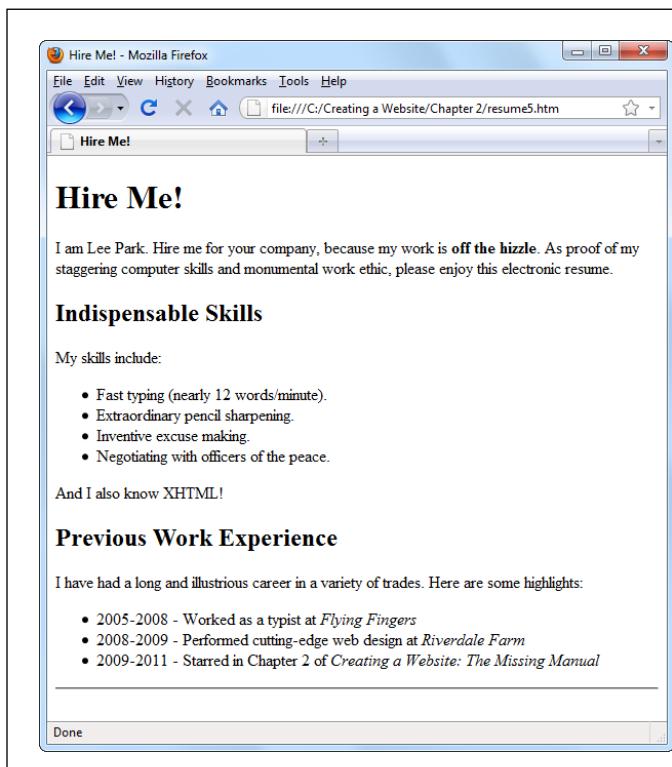
<h2>Previous Work Experience</h2>
<p>I have had a long and illustrious career in a variety of trades. Here are some highlights:</p>
<ul>
    <li>2005-2008 - Worked as a typist at <i>Flying Fingers</i></li>
    <li>2008-2009 - Performed cutting-edge web design at <i>Riverdale Farm</i></li>
    <li>2009-2011 - Starred in Chapter 2 of <i>Creating a Website: The Missing Manual</i></li>
</ul>

<hr />
</body>

</html>

```

Figure 2-13 shows this revised version of the web page.



The screenshot shows a Mozilla Firefox window titled "Hire Me! - Mozilla Firefox". The address bar displays "file:///C:/Creating a Website/Chapter 2/resume5.htm". The main content area of the browser shows the following HTML document:

```

<h1>Hire Me!</h1>
<p>I am Lee Park. Hire me for your company, because my work is off the hizzle. As proof of my staggering computer skills and monumental work ethic, please enjoy this electronic résumé.</p>
<h2>Indispensable Skills</h2>
<p>My skills include:</p>


- Fast typing (nearly 12 words/minute).
- Extraordinary pencil sharpening.
- Inventive excuse making.
- Negotiating with officers of the peace.
- 2005-2008 - Worked as a typist at Flying Fingers
- 2008-2009 - Performed cutting-edge web design at Riverdale Farm
- 2009-2011 - Starred in Chapter 2 of Creating a Website: The Missing Manual


Figure 2-13:  

Featuring more headings, lists, and a horizontal line, this HTML document adds a little more style to the résumé.


```

POWER USERS' CLINIC

Plug-Ins: Going Beyond HTML

The creators of HTML designed it perfectly for putting research papers and other unchanging documents on the Web. They didn't envision a world of Internet auctions, e-commerce shops, or browser-based games. To make this work, enterprising coders have harnessed a slightly quirky language called JavaScript (described in Chapter 15), which works hand-in-hand with HTML. JavaScript is right at home performing bread-and-butter tasks like collecting user information and sending it to the web server, or refreshing a page with new information. But more ambitious tasks—say, providing an interactive world map or a 3-D product model—are incredibly difficult to achieve or outright impossible (even if you work at Google).

To transcend limits like these, crafty technology companies (like Microsoft, Google, Apple, and Adobe, to name just a few) have created browser plug-ins. A *plug-in* is essentially a miniature program that runs right inside your web browser. For example, if you have the Adobe PDF plug-in, you can view an Adobe PDF document in your browser window. More impressive plug-ins, like Microsoft Silverlight and Adobe Flash, allow your browser to run rich web applications that show animations (say, for advertisements), play videos (as on YouTube), and even create interactive arcade games (see www.ferryhalim.com/orisinal for examples of some addictive Flash games).

You can often identify whether something other than HTML is at work by right-clicking the suspected content region in a page. For example, right-click a Flash region and a pop-up menu appears with options to configure Flash or get additional information. If you don't have the Flash plug-in but need it for a page, the page will ask if you want to download the latest version of the plug-in. (You'll use Flash to put a slick music player in your web page in Chapter 17.)

Although plug-ins are impressive and can do far more than ordinary HTML, they run into compatibility headaches, especially in the ever-expanding world of web-connected devices. For example, even though Flash is the most popular plug-in in existence (installed on some 99% of web surfing computers), Apple's iPad, iPhone, and iPod Touch do not currently support it.

This limitation is part of the reason that web developers and browser companies are eagerly anticipating HTML5 and a new set of web application standards. Although many of these emerging technologies are still far in the future, eventually they will make it possible for black-belt programmers to build rich web applications without leaving HTML behind.

Checking Your Pages for Errors

Even a web designer with the best intentions can write bad markup and break the rules of HTML. Although browsers really *should* catch these mistakes, virtually none of them do. Instead, they do their best to ignore mistakes and display flawed documents.

At first glance, this seems like a great design—after all, it smoothes over any minor slip-ups you might make. But there's a dark side to tolerating mistakes. In particular, this behavior makes it all too easy for serious errors to lurk undetected in your web pages. What's a serious error? A problem that's harmless when you view the page in your favorite browser, but makes an embarrassing appearance when someone views the page in another browser; a mistake that goes undetected until you edit the code, which inadvertently exposes the problem the next time your browser displays the

page; or an error that has no effect on page display but prevents an automated tool (like a search engine) from reading the page.

Fortunately, there's a way to catch problems like these. You can use a *validation tool* that reads through your web page and checks its markup. If you use a professional web design tool like Dreamweaver, you can use its built-in error checker (Chapter 4 has the details). If you create pages by hand in a text editor, you can use a free online validation tool.

Here are some potential problems that a validator can catch:

- Missing mandatory elements (for example, the `<title>` element)
- A start tag without a matching end tag
- Incorrectly nested tags
- Tags with missing attributes (for example, an `` element without the `src` attribute)
- Elements or content in the wrong place (for example, text that's placed directly in the `<head>` section)

You can find plenty of validation tools online. Here's how to use the popular validator provided by the W3C standards organization:

1. **Make sure your document has the right doctype.**

Your doctype tells the validator what rules to use when validating your document. For example, if you use the HTML5 doctype (page 31), the validator checks that your document meets the rules of HTML5 and allows some features that aren't acceptable in the older XHTML 1.0 standard.

Note: Because the HTML5 standard is still evolving, HTML5 validators aren't yet as polished as their XHTML counterparts. For example, at the time of this writing the W3C's HTML5 validator is considered to be "experimental."

2. **In your web browser, go to <http://validator.w3.org> (Figure 2-14).**

The W3C validator gives you three choices, which are represented by three separate tabs: Validate by URI (for a page that's already online), Validate by File Upload (for a page that's stored in a file on your computer), and Validate by Direct Input (for markup you type directly into the provided box).

3. **Click the tab you want, and supply your HTML content.**

Validate by URI lets you validate an existing web page. Simply enter the URL for the web page in the Address box (like `www.MySloppySite.com/Flawed Page.html`).

Validate by File Upload lets you upload any file from your computer. First, click the Browse button (which is actually called Choose File in Chrome) to show the standard Open dialog box. Then, browse to the location of your HTML file, select it, and then click Open.

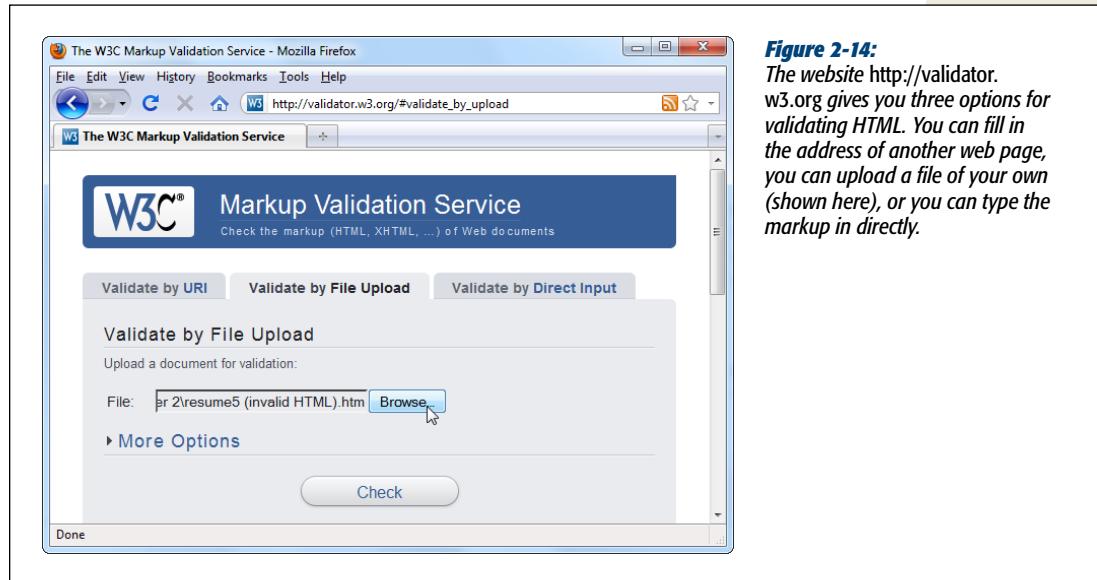


Figure 2-14:
The website http://validator.w3.org/#validate_by_upload. w3.org gives you three options for validating HTML. You can fill in the address of another web page, you can upload a file of your own (shown here), or you can type the markup in directly.

Validate by Direct Input lets you validate any markup—you just need to type it into a large box. The easiest way to use this option is to copy the markup from your text editor and paste it into the box on the W3C validation page.

Before continuing you can click More Options to set other options, but you probably won't. It's best to let the validator automatically detect the document type; that way, the validator will use the doctype specified in your web page. Similarly, use automatic detection for the character set unless your HTML page isn't in English and the validator has trouble determining the correct character set.

4. Click the Check button.

C sends your HTML page to the W3C validator. After a brief delay, the validator reports whether your document passed the validation check or, if it failed, what errors the validator detected (see Figure 2-15).

The validator also may offer a few harmless warnings for a perfectly valid HTML document, including a warning that the character encoding was determined automatically and a warning that the HTML5 validation service is considered to be an experimental, unfinished feature.

The screenshot shows a Mozilla Firefox window displaying the results of a W3C Markup Validator check on a file named "resume5 (invalid HTML).htm". The title bar of the browser window reads "[Invalid] Markup Validation of resume5 (invalid HTML).htm - W3C Markup Validator - Mozilla Firefox". The address bar shows the URL "http://validator.w3.org/check". The main content area of the browser displays the validation output.

Validation Output: 10 Errors

- Line 7, Column 7: Element head is missing a required instance of child element title.**
Content model for element `head`:
If the document is an `iframe_srcdoc` document or if title information is available from a higher-level protocol: Zero or more elements of `metadata_content`.
Otherwise: One or more elements of `metadata_content`, of which exactly one is a `title` element.
`</head>`
- Line 18, Column 53: End tag li seen, but there were open elements.**
`Fast typing (nearly 12 words/minute).`
- Line 18, Column 11: Unclosed element b.**
`Fast typing (nearly 12 words/minute).`

Done

Figure 2-15:
In this file, the validator has discovered 10 errors that stem from two mistakes. First, the page is missing the mandatory `<title>` element. Second, it closes the `` element before closing the `` element nested inside. (To solve this problem, you would replace `` with ``.) Incidentally, this file is still close enough to correct that browsers can display it correctly.

Putting Your Page on the Web

In the previous chapter you learned the basics of HTML by creating a simple one-page résumé. You can do a lot more to perfect that page, but before going any further, pause a moment to consider one of the most important pieces of the web puzzle: getting your pages online.

In this chapter, you'll learn how web servers work and how you can put them to work for you. Once you're armed with these high-tech nerd credentials, you'll be ready to search for your own *web host*, a company that lets you park your site on its web server. All you need to do is figure out your requirements, see which hosts offer what you need, and start comparison shopping.

How Web Hosting Works

As you learned in Chapter 1, the Web isn't stored on any single computer, and no company owns it. Instead, the individual pieces (websites) are scattered across millions of computers (web servers). Only a subtle illusion makes all these sites seem part of a single environment. In reality, the Internet is just a set of standards that let independent computers talk to each other.

So how does your favorite browser navigate this tangled network of computers to find just the web page you want? It's all in what's known as the *URL* (Uniform Resource Locator)—which is simply the website address you type into your browser, like www.google.com.

Understanding the URL

A URL consists of several pieces. Some are optional, because a browser or web server can fill in some blanks automatically. Others are required. Figure 3-1 dissects the URL <http://www.SellMyJunkForMillions.com/Buyers/listings.htm>.



Web addresses pack a lot of information into a single line of text:

- **The protocol** indicates your chosen method of transmission—in other words, how your browser should communicate with the web server. Websites always use HTTP (HyperText Transport Protocol), which means the protocol portion of a website URL is always *http://* or *https://*. (The latter establishes a super-secure connection over HTTP that encrypts sensitive information you type, like credit card numbers or passwords.) In most browsers, you can get away without typing in this part of the URL. For example, when you type *www.google.com*, your browser automatically converts it to the full URL, *http://www.google.com*.

Note: Although *http://* is the way to go when browsing the Web, depending on your browser, you may use other protocols for other tasks. Common examples include *ftp://* (File Transfer Protocol) for uploading and downloading files and *file:///* for retrieving a file directly from your own computer's hard drive.

- **The domain** name identifies the server that hosts the site you want to see. By convention, server names usually start with *www* to identify them as World Wide Web servers. In addition, as you'll discover later in this chapter, friendly-seeming domain names like *www.google.com* or *www.microsoft.com* are really just stand-ins for what your browser really needs in order to locate a server—namely, its numeric address (see page 52).
- **The path** identifies the folder where the server stores the specific web page you're looking for. This part of the URL can have as many levels as needed. For example, the path */MyFiles/Sales/2011/* refers to a *MyFiles* folder that contains a *Sales* folder that, in turn, contains a folder named *2011*. Windows fans, take note—the slashes in the path portion of a URL are ordinary forward slashes, not the backward slashes used in Windows file paths (like *c:\MyFiles\Current*). This

convention matches the file paths Unix-based computers use, which were the first machines to host websites. It's also the convention used in modern Macintosh operating systems.

Note: Many browsers are smart enough to correct the common mistake of typing in the wrong type of slash. However, you shouldn't rely on this because similar laziness can break the web pages you create. For example, if you use the element to link to an image and use the wrong type of slash, your picture won't appear.

- **The file name** is the last part of the path and it identifies the specific web page you're requesting. Often, you can recognize it by the file extension .htm or .html, both of which stand for HTML.

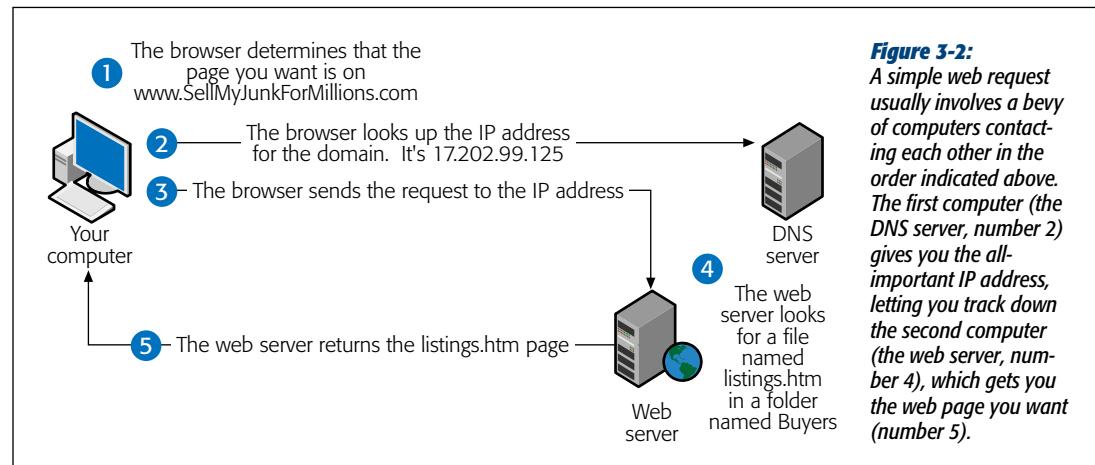
Note: Web pages often end with .htm or .html, but they don't need to. For example, if you look at a URL and see the strange extension *.blackpudding*, odds are you're still looking at an HTML document. In most cases, browsers ignore an extension as long as the file contains information that a browser can interpret. However, just to keep yourself sane, this is one convention you shouldn't break.

- **The fragment** is an optional part of a URL that identifies a specific position within a web page. You can recognize a fragment because it always starts with the number-sign character (#) and appears after a file name. For example, the URL <http://www.LousyDeals.com/index.html#New> includes the fragment #New. When you click the URL, it takes you to the section of the *index.html* page where the page creator has placed the New bookmark. You'll learn how to place bookmarks on page 229.
- **The query string** is an optional part of a URL that some websites use to send extra information from one web page to another. You can identify the query string because it starts with a question mark (?) and appears after a file name. To see a query string in action, go to Microsoft's search engine www.bing.com and perform a search for "pet platypus." When you click the Search button, you're directed to a URL like http://www.bing.com/search?q=pet+platypus&form=QB_LH&filt=all&qs=n. This URL is a little tricky to analyze, but if you hunt for the question mark, you'll discover that you're on a page named "search." The information to the right of the question mark indicates that you're executing an English-language search for pages that match both the "pet" and "platypus" keywords. When you request this URL, a specialized Microsoft search program analyzes this query string to determine what type of search it needs to perform.

Note: You won't use the query string in your own web pages, because it's designed for heavy-duty web programs like the one that powers Bing (or Google). By understanding the query string, however, you gain a bit of insight into how other websites work.

How Browsers Analyze a URL

Clearly, a URL packs a lot of information in one place. But how does a browser actually use the URL to fetch the web page you want? To understand that, take a peek behind the scenes (see Figure 3-2).



For example, after you type <http://www.SellMyJunkForMillions.com/Buyers/listings.htm> into the address bar and press Enter, here's what happens:

1. The browser figures out what web server to contact by extracting the domain name from the URL.

In this example, the domain name is www.SellMyJunkForMillions.com.

2. To find the server named www.SellMyJunkForMillions.com, the browser converts the domain name into a computer-friendly number called the **IP address**.

Every computer on the Web—from web servers to your own machine—has its own unique IP address. To find the IP address for a server, the browser looks up the server's domain name in a giant online catalog called the DNS (Domain Name Service). An IP address looks like a set of four numbers separated by periods (or, in techie speak, dots). For example, the www.SellMyJunkForMillions.com website may have the IP address 17.202.99.125.

Note: The DNS catalog isn't stored on your computer, so your browser gets this information from the Internet. You can see the advantage of this approach. Under ordinary circumstances, the domain name for an online business never changes; it's the public face customers use and remember. Behind the scenes, however, its an IP address may change, because the business moved its website from one server to another. As long as the business updates the DNS, the move won't cause any disruption. Fortunately, you won't need to worry about managing the DNS yourself if it moves to a new server, because the company that hosts your site automatically handles that for you.

3. The browser sends the page request to the web server's now-retrieved IP address.

The actual route the message takes is difficult to predict. It may cross through a number of other servers on the way.

4. The server receives the request and looks at the path and file name in the URL.

In this case, the server sees that the request is for a file named *listings.htm* in a folder called *Buyers*. The server looks up that file, and then sends it back to the requesting web browser. If the file doesn't exist, the server sends back an error message.

5. The browser receives the HTML page it was been waiting for (*listings.htm*) and then displays that page for your viewing pleasure.

The URL <http://www.SellMyJunkForMillions.com/Buyers/listings.htm> is a typical example. In the wild, however, you'll sometimes come across URLs that seem a lot simpler. For instance, consider <http://www.amazon.com>. It clearly specifies the domain name (www.amazon.com), but it doesn't include any information about the path or file name. So what's a web server to do? It sees that you aren't requesting a specific file, so it sends you the site's fall-back page—its *default* page, in geekspeak, which is often named *index.htm* or *index.html*, and is what you know as the site's home page. (However, a web administrator can configure any page as the default.)

Now that you understand how URLs work, you're ready to integrate your own pages into the fabric of the Web. The first task is getting yourself a good domain name.

POWER USERS' CLINIC

Internet vs. Intranet

As you already know, the Internet is a huge network of computers that spans the globe. An *intranet* is a lot smaller; it's a network inside a specific company, organization, or home that joins together a much smaller number of computers. In fact, an intranet could have as few as two computers.

An intranet makes sense anytime you need a website available to only a small number of people in one location. For example, a company can use an intranet to share web pages that have marketing information (or the latest office gossip). In your own home, you could let your housemates browse your web creations from their own computers. The only limitation is that a website on an intranet is accessible to the computers on that network only. Other web visitors won't be able to see it.

Setting up a website for an intranet is easier than setting one up for the Internet because you don't need to register a domain name. Instead, you can use the network computer's name. For example, if your computer has the network name SuperServer, you could access a web page with a URL like <http://SuperServer/MySite/MyPage.htm>. Of course, this works only if you've set up a home network (easy) and some server software (a little more complicated). For example, on a Windows computer, you might make use of the IIS (Internet Information Services) software, which most versions of Windows include, but you need to explicitly enable and configure the program. These tasks are outside the scope of this book. To learn more, consult a good networking resource.

Domain Names

Shakespeare may have famously written “What’s in a name? That which we call a rose/By any other name would smell as sweet.” But he may have seen things differently if he had to type *www.thesweetsmellingredflowerwiththorns.biz* into his browser instead of *www.rose.com*. Short, snappy domain names attract attention and are easy to remember. Today, cheap, personalized domain addresses are within the reach of every website creator. If you decide to get one of your own, take the time to get it right.

Note: Valid domain names include only letters, numbers, and dashes.

Choosing the Right Domain Name

You’ll find that most short, clever word combinations are no longer available as domain names. Even if they aren’t in use, *domain squatters*—individuals who buy and hold popular names in hopes of selling them to desperate high bidders later—have long since laid claim to common names. Give up on *www.worldsbestchocolate.com*—it’s gone. However, you may find success with names that are a little longer or more specific (*www.worldsbestmilkchocolate.com*), use locations or the names of people (*www.bestvermontchocolate.com* or *www.anniesbestchocolate.com*), or introduce made-up words (*www.chocolattech.com*). All these domain names are available at the time of this writing.

Choosing a good domain name is not an exact science, but you can find plenty of anecdotal evidence on names that don’t work. Here are some mistakes to avoid:

- **Too-many-dashes.** It may be tempting to get exactly the domain name you want by adding extra characters, like dashes, between words. For example, you have no chance at getting *www.popularbusiness.com*, but *www.popular-business.com* is still there for the taking. Think carefully. Dashes can confuse some people, and others may overlook them. (Other characters, like underscores, are far worse—avoid them at all costs.) Some webmasters believe that a domain name with a single dash is perfectly reasonable, but one with several dashes looks like a spam site, and should be avoided.
- **Phrases that look confusing in lowercase.** Domain names aren’t case-sensitive, and when you type a domain name into a browser, the browser converts everything to lowercase. The problem is that some phrases can blend together in lowercase, particularly if you have words that start with vowels. Take a look at what happens when the documentation company Prose Xact puts their business name into a lowercase domain name: *www.prosexact.com*. You get the picture. (Incidentally, this is one situation where you might want to resort to a dash.)

Tip: Even though domain names don't distinguish case, that doesn't stop businesses from using capital letters in business cards, promotions, and marketing material to make the domain name more readable. Whether customers type *www.google.com* or *wWw.gOOgLE.cOm* into their browsers, they get to the same site.

- **Names that don't match your business.** It's a classic business mistake. You set up a flower shop in New York called Roses are Red. Unfortunately, the domain *www.rosesarerered.com* is already taken so you go for the next best choice, *www.newyorkflorist.com*. Huh? What you've actually done is create two separate business names, and a somewhat schizophrenic identity for your company. To avoid these problems, if you're starting a new business, try to choose your business name and your domain name at the same time so they match. If you already have a business name, settle on a URL that has an extra word or two, like *www.rosesarereredflorist.com*. This name may not be as snappy as *www.newyorkflorist.com*, but it avoids the inevitable confusion of creating a whole new identity.
- **Unusual top-level domains.** The last few letters of the domain (the part after the last period) is called the *top-level domain*. Everyone wants a .com address for their business, and as a result they're the hardest domain name to get. Of course, there are other top-level domains, like .net, .org, .biz, .info, and .name. The problem is, most web visitors expect a .com. If you have the domain name *www.SuperShop.biz*, odds are someone will type in *www.SuperShop.com* while trying to find your site. That mistake can easily lead your fans to a competitor (or to a vastly inferior website). In other words, it's sometimes worth taking a second-choice name to get your first choice of a top-level domain (a .com).

Note: The top-level domain .org was originally intended for nonprofit organizations. It's now available for anyone to use and abuse. However, if you're setting up a nonprofit of your own, the .org domain makes more sense than .com and is just as recognizable.

In the time since the first edition of this book, it's become even harder to get a decent domain name. In the past, your only competition was other people planning to set up a website and unscrupulous domain name resellers looking to buy a hot name and flip it for a big profit. But now, nefarious people buy just about any domain name at the drop of a hat, build a fly-by-night web page filled with ads, and wait a few months to see how much unsuspecting web traffic stumbles their way. This practice is called *domain tasting*, and it's surprisingly profitable. The bottom line? It's still possible to cook up a decent domain name that's still available, but you'll need a dash of compromise and all the creativity you've got.

Note: There are a pile of new top-level domains that are planned but not yet available for people to use. They include .eco, .love, .god, .sport, .gay, and just about anything else applicants can get their hands on. These domains won't start appearing until 2012, at the earliest.

FREQUENTLY ASKED QUESTION

International Domain Names

Some domain names end with a country code. Should I get one?

A .com address is a website creator's best friend. But if you can't get the .com you need, another reasonable option might be to go with a country-specific top-level domain like .us (USA) or .ca (Canada).

For example, if you're offering piano lessons in England, www.pianolessons.co.uk isn't a bad choice. If you're planning to sell products to an international audience, however, www.HotRetroRecords.co.uk could frighten away otherwise interested buyers, who may assume dealing with a British seller is too much trouble.

Special rules apply regarding who can register country-specific names. For example, some are only available to people who live in the corresponding country or who have

a registered business there. Due to these restrictions, some web hosting companies can't sell certain country-specific domains. If you have trouble registering the country-specific name you want, you can use Google to find a registrar that supports your choice. For example, to find a registrar for Australian domains, search for "Australian domain names."

Each country in the world has its own unique top-level domain. But some of them are now available for other people to use in more creative ways. For example, the top-level domain .tv was created for the tiny country of Tuvalu, but it's now available for anyone who wants to create a television-focused website (with the Tuvaluan government getting a small cut of the profits). Similarly repurposed domains include .me, .cd, .tm, and .ws. You can find more information about these odder domain name choices on Wikipedia, at <http://tinyurl.com/rnlmf>.

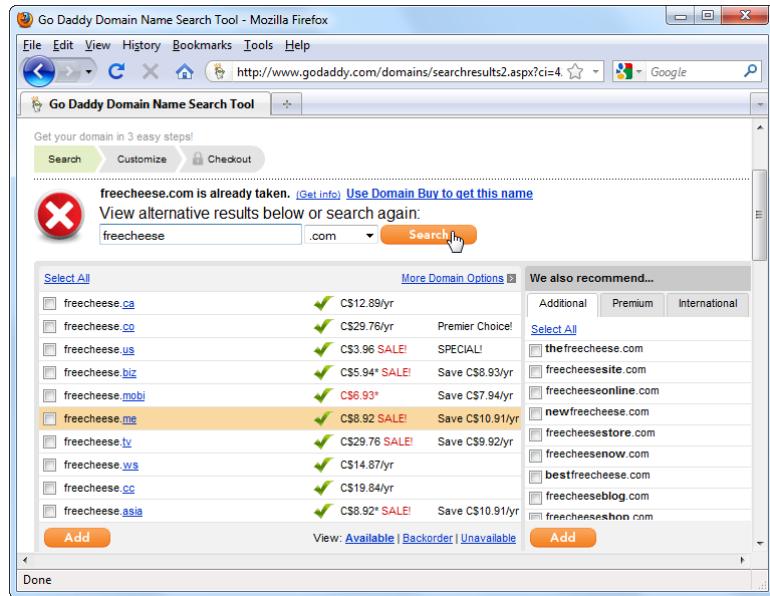
Searching for a Name

With a few domain name ideas in hand, start checking their availability. You can do this even if you haven't chosen a web host. In fact, the Web abounds with tools that let you check availability of a domain name.

Tip: Domain name searches are an essential bit of prep work. Try to come up with as many variations and unusual name combinations as possible. Aim to find at least a dozen available names so you give yourself lots of choice. Once you compile the list, why not make a few late-night phone calls to pester friends and relatives for their first reactions?

Just about every web hosting company provides its own version of a domain name search tool. Figure 3-3 shows one from www.godaddy.com. To get started, type in the domain you want most and click Search. (Notice that in most cases you don't have to type in the *www* at the start of the domain name or the .com at the end, because those parts are assumed.)

When you perform a search and find an available domain name, the hosting company gives you the option to buy it. But don't do anything yet, because you still need to find the best host.

**Figure 3-3:**

This search reveals that your first choice, www.freecheese.com, is gone. All that's left are the less-catchy www.freecheese.biz and country-specific domains. The 'Additional' list on the right generates a few similar domain names that are available (in this case, that includes www.thefreecheese.com and www.freecheesosite.com). You can also click the Premium link to see domain names for sale by their owners; be prepared to shell out some serious bank, however (like www.fatfreecheese.com for over \$1,000 and www.cheesenow.com for more than \$3,000).

Note: You may think you could see if a domain is free just by typing it into your web browser. But this method of checking takes more time, and it doesn't give you a definitive result. Someone can buy a domain name without setting up a website, so even if you can't find a site using your browser, the domain may not be available.

Registering Your Name

After you find an available name, you probably want to wait to register it until you're ready to sign up for a web hosting plan, too (which you'll read about in the next section). Most web hosts offer free or discounted domain name registration when you rent space from them. In addition, doing both at once is the easiest way to set up your domain name, because the process automatically establishes a relationship between your domain name and your website.

In a few cases, however, you may want to register a domain name separately from your web hosting package. Here are some examples:

- You don't want to create a website right now, but you do want to register a name so no one else can grab it (a tactic known as *domain parking*). Sometime in the future, you may develop a website that uses that name.

- You already have web space, possibly through your ISP (Internet Service Provider). All you need to make your website seem more professional is to get a personalized domain name. This option can get a little tricky, and you may need to use a procedure called *domain forwarding* (which you'll read about in a moment).
- Your hosting company can't register the type of domain you want. This can happen if you need a domain name with a country-specific top-level domain.

If you don't fall into one of these special categories, skip ahead to the section "Getting Web Space" (page 62) to start searching for the right web host. Otherwise, keep reading for details on registering and managing a domain name on its own.

Note: All web hosting companies let you register more than one domain for the same website. That means you can register both *www.FancyPants.com* and *www.FancyPants.biz*, and specify that both these addresses point to the same website. Of course, you'll need to pay an extra domain name registration fee. (Really big web companies use this strategy to accommodate typos potential visitors make. For instance, see where *amzn.com* and *googel.com* take you.)

FREQUENTLY ASKED QUESTION

A Host Here, a Domain There

Can I buy my domain name and web space from different companies and still make them work together?

The simplest approach is to get both from the same company, but that's not always possible. Maybe you bought your domain name before you set up your website and you don't want to pay the cost of transferring the domain. Or maybe you have a country-specific domain name (like *www.CunningPets.co.uk*) that your web host can't register. Or maybe you just want the flexibility to change web hosts frequently, so you can get the best service or cheapest rates.

To make this multiple-company tango work, you need some technical support from your web hosting company. Contact their help desk and let them know what you plan to do. They can give you specific instructions, and they'll set up their name servers (more on what those are in a moment) with the right information for your domain.

The next task is to change the registration information for your domain. To do so, follow these steps:

1. Find out the name of the *domain name servers (DNS servers)* at your web hosting company. These are the

computers that convert domain names into numeric IP addresses (page 52). The technical support staff can give you this information.

2. Go to the company where you registered the domain name and update your domain registration settings. Change the name server setting to match the name servers you found out about in step 1 (as shown in Figure 3-4).

Due to the way DNS servers work, the change can take 24 hours or more to take effect.

When you make this change, you're essentially saying that your web host is now responsible for giving out the IP address of your website. When someone types your domain name into a browser, the browser will contact the name server at your hosting company to get the IP address. From that point on, it's smooth sailing.

Once you modify your domain name registration, you still have the same two bills to pay. You'll pay your hosting fees to the web hosting company and the yearly domain name registration fee to the company where you registered your domain name.

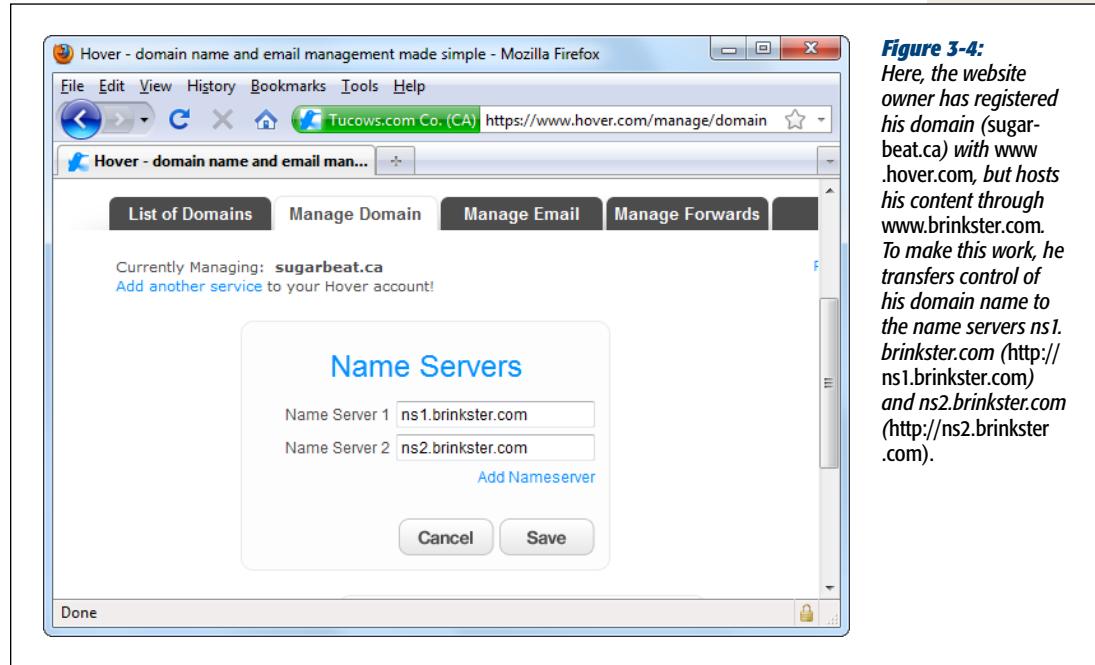


Figure 3-4: Here, the website owner has registered his domain (`sugarbeat.ca`) with `www.hover.com`, but hosts his content through `www.brinkster.com`. To make this work, he transfers control of his domain name to the name servers `ns1.brinkster.com` (`http://ns1.brinkster.com`) and `ns2.brinkster.com` (`http://ns2.brinkster.com`).

Domain parking

Domain parking means you've registered a domain name but haven't yet purchased any other services, like renting web storage space.

Most people use domain parking to put a domain name away on reserve. In the increasingly crowded world of the Web, many people use it to protect their own names (for example, `www.samanthamalone.com`). Domain parking is also useful if you want to secure several potential business names you may use in the future.

Tip: If you do reserve a domain name, do your research and pick a company that you'd also like to use to host your website as well. You can switch domain names from one web host to another, but it's a bit of a pain. Contact the host you're currently working with for specific instructions.

The real appeal of domain parking is that it's cheap. You pay a nominal domain-name registration fee (as little as \$5 a year), and you get to keep the name for as long as you're willing to pay for it.

Domain forwarding

Domain forwarding lets you combine a personalized domain name with web space you already have. For example, you may have free web space through your ISP, your school, your job, from a free web hosting service, or from a crazy uncle with a web

server in his basement. In these situations, you can save some money because you don't need to pay a company to host your site. However, you don't get to pick your domain name. For example, if you have web space on an ISP, you might be stuck with a URL like <http://member.magicisp.com/members/personalwebspace/~henryj420/home>, which clearly isn't as catchy as www.HenryTheFriendly.com.

Take heart. Even if you get free web hosting, you can still give your website a snappy URL. The trick is to buy the URL separately and use domain forwarding to point your brand-new URL to your existing website. In the previous example, for instance, you could buy the domain name www.HenryTheFriendly.com and use domain forwarding to point www.HenryTheFriendly.com to your web space on <http://member.magicisp.com>.

Tip: Even if the URL from your ISP isn't as bad as in the example above, buying your own domain name is still a good idea. Here's why: If you use an ISP-provided URL and your ISP changes its configuration or you switch from one ISP to another, your web visitors won't be able to find your site anymore. But if you use your own URL and domain forwarding, you simply need to update your domain-name settings to reflect your ISP's new configuration or your new ISP itself. All the while, your custom domain name keeps working. No one will even notice the change.

To use domain forwarding, you first need to register a domain name that comes with forwarding as an included service. Then you can log in and set the forwarding settings (see Figure 3-5).

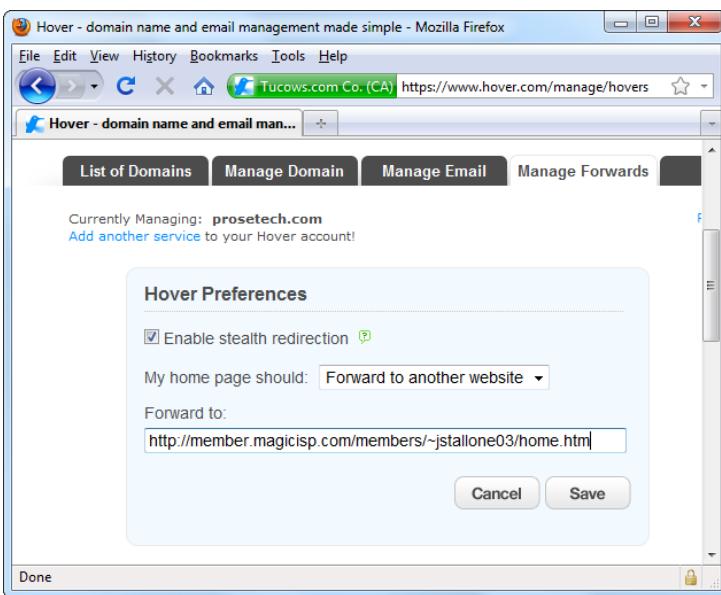


Figure 3-5:

Here, a domain-forwarding service sends visitors who type in www.HenryTheFriendly.com to a much more awkward URL where the website is actually located (bottom box). Usually, web hosts implement domain forwarding in such a way that the browser address bar displays the original domain name even though the host sends visitors to a different site.

Most hosts that offer domain forwarding let you *mask* your URL (see the checkbox in Figure 3-5, for instance). If you do so, visitors will have no idea that your host redirected them because the URL in their browser didn't change. So if you send your visitors from www.HenryTheFriendly.com to some long, unwieldy URL, the original address, <http://www.HenryTheFriendly.com>, stays in the address bar. Not only does masking give your site a cleaner, more professional look, it also ensures that if a visitor bookmarks your site, the bookmark gets the original URL (the domain name you bought), not the redirected one (where you host your web content).

On the other hand, if you use domain forwarding to send visitors from an alternate website (like a .biz site) to your main website, you shouldn't mask the URL. In this situation, it makes more sense to have your main site's URL in the browser's address bar.

Note: URL masking is sometimes called *stealth redirection*.

However, domain forwarding is a bit of a patchy trick, however, and sometimes the cracks show through. One problem is that domain forwarding can confuse search engines. For example, when your website appears in a Google search, it might show your real website URL (the one you're forwarding to), rather than the domain name you purchased. Domain forwarding can also cause headaches when tracking your visitors. Everybody who surfs into your website will appear to come from your forwarded domain, and you'll lose the information about how they reached your website in the first place (for example, whether they came from a link on another website or through Google). And finally, URL forwarding adds a slight, almost imperceptible delay to your page, because when people visit your website their browsers need to talk to two different web servers.

Free Domain Names

As you no doubt know, the Web is a great place for frugal shoppers. Not only can you score a great deal on a sporty iPod and a used sofa bed, you can also pick up a web domain name for the princely sum of zero dollars. The trick is to register at a free subdomain service.

Subdomains are extensions that build on an existing domain name. A free subdomain service might own a domain like www.net.tc and let you register a subdomain like www.acebusiness.net.tc. Sure, the end portion of the domain (.net.tc) is a little awkward, but hey—it's free. Even better, getting popular words in a subdomain is much easier than getting them in a standalone domain name. For example, don't bother trying to scoop up www.chocolates.com or the .net, .org, .biz, .us, .ca versions of the name, as they're all taken. But if you head over to popular subdomain service www.smartdots.com (Figure 3-6), you can still register www.chocolates.net.tc without opening your wallet.

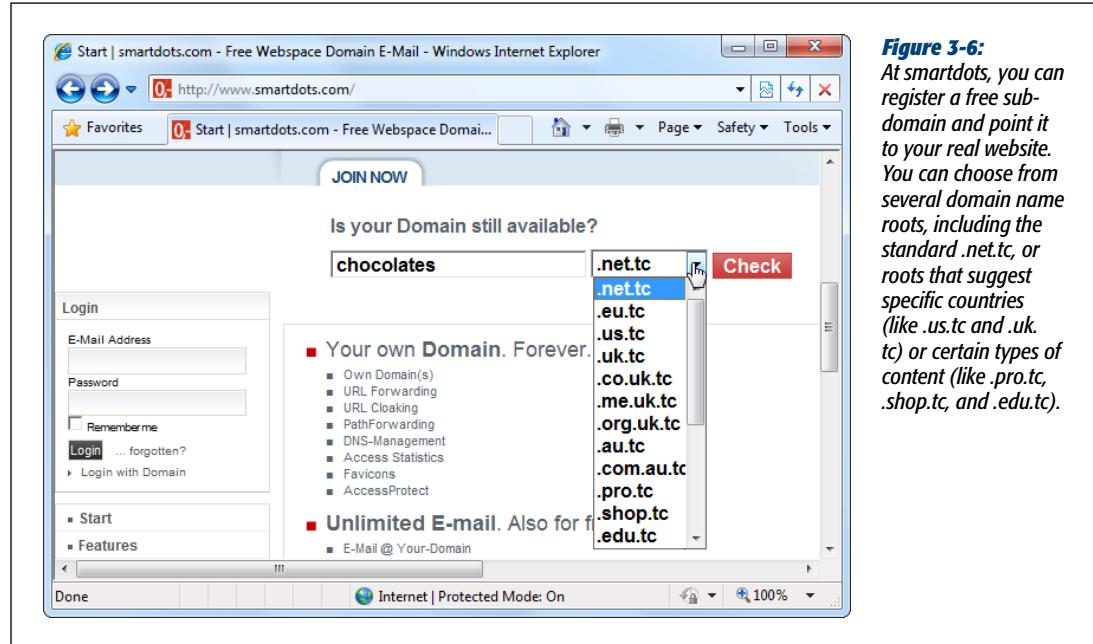


Figure 3-6:
At smartdots, you can register a free sub-domain and point it to your real website. You can choose from several domain name roots, including the standard .net.tc, or roots that suggest specific countries (like .us.tc and .uk.tc) or certain types of content (like .pro.tc, .shop.tc, and .edu.tc).

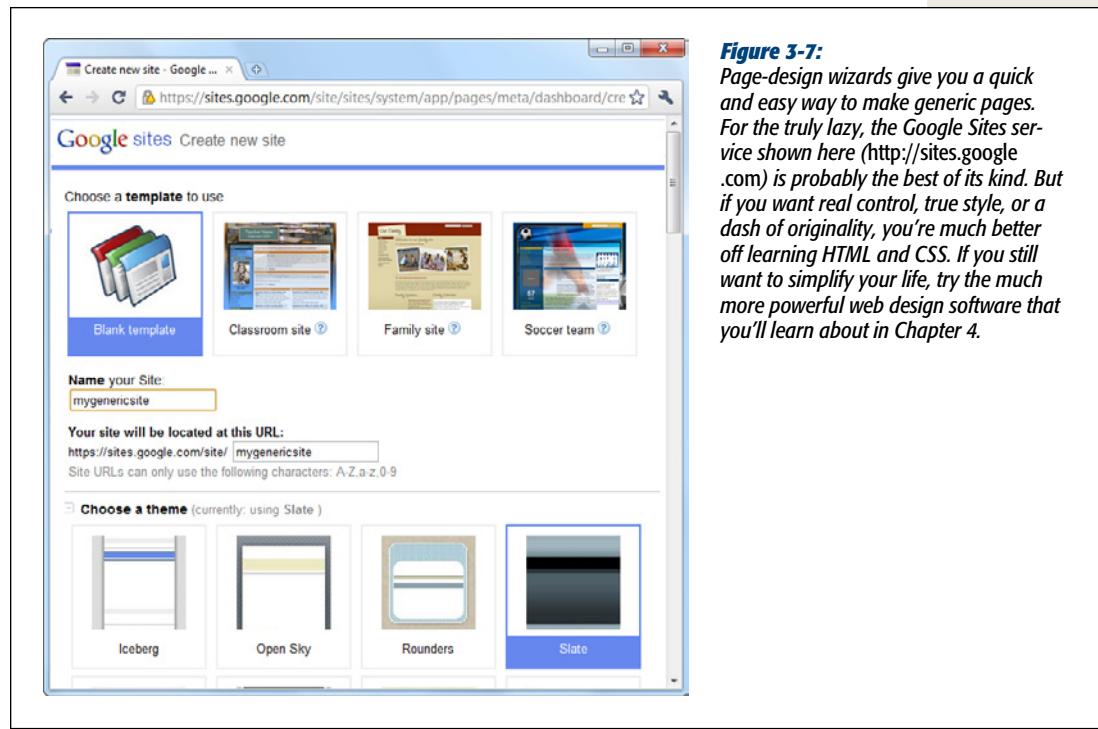
Getting Web Space

All you need to achieve web superstardom is a domain name and a small amount of space on a web server. There's no one-size-fits-all solution when it comes to finding a web host. Instead, you choose a hosting company based on your budget, what you want your website to do, and your own capricious whims (let's face it—some hosting companies just have way cooler names than others).

What you *don't* want is a web host that offers some sort of special software that promises to help you create a website in two or three easy steps (see Figure 3-7). These tools range from mediocre to terrible. After all, if you're content to create the same cookie-cutter website as everyone else, you probably aren't interested in learning HTML, and you wouldn't have picked up this book.

Instead, you want standard website hosting. Here, you're given a slot of space on a server to manage as you see fit. You create your pages on your own computer, and then copy these files to the server so others can view them. This type of web hosting is all you need to use this book.

Web hosting packages usually charge a monthly fee. For basic hosting, this fee starts at the reasonable sum of \$5 to \$10 per month. Of course, that cost can escalate quickly, depending on the features you want your host to provide.

**Figure 3-7:**

Page-design wizards give you a quick and easy way to make generic pages.

For the truly lazy, the Google Sites service shown here (<http://sites.google.com>) is probably the best of its kind. But if you want real control, true style, or a dash of originality, you're much better off learning HTML and CSS. If you still want to simplify your life, try the much more powerful web design software that you'll learn about in Chapter 4.

Assessing Your Needs

Before you decide on a host, ask yourself one important question—what features do you need? Web hosts are quick to swamp their ads and websites with techie jargon, but they don't tell you which services are truly useful. Here's a quick overview that describes what hosts sell and what you need to know about each offering. If you'd like to keep track of which features you need, you can use the checklist on page 65.

- **Web space** is how much server space you rent to store your website. Although HTML pages are extremely small, you may need more space to fit in images or files you want others to download, like a video of your wedding. A modest site can easily survive with a measly 100 MB (that's megabytes, not gigabytes) of space, unless you're stuffing it full of pictures or videos. Most web hosts throw in 10 or 100 times more server space, knowing you'll probably never use it.

Note: For the numerically challenged, a gigabyte (GB) is the same as 1,024 megabytes (MB). To put that in perspective, today's hard drives can have 500 GB of space or more, enough room for tens of thousands of websites.

- **Bandwidth** (or web traffic, as it's sometimes called) is the maximum amount of information you can deliver to anyone who visits your site in a month. Usually, you can make do with the lowest bandwidth your hosting company offers. For more information, see the box on page 65.
- A **domain name** is a custom website address, as in *www.HenryTheFriendly.com*. If you decide to get a personalized domain name, you don't necessarily need to get it from the same company that hosts your site. Getting both from the same source, however, does make life easier, and hosting companies often throw in one or more domain names at a discounted price when you sign up for a hosting plan.
- **Email addresses.** Odds are, you already have some of these. But you may want an email address that matches your website address, especially if you're paying for a customized domain name. For example, if you own *www.HenryTheFriendly.com*, you'd probably like to use an email address like *Hank@HenryTheFriendly.com*. Most hosting companies let you create dozens or hundreds of email addresses at your domain (which is good if you're running a small company), and most also let you read your email in your browser or send it to a desktop program like Microsoft Outlook.
- **Upload-ability.** The ease of transferring files to your server is another important detail. As you saw in the previous chapter, you can perfect your web pages on your own computer before you upload them. But once your website is ready for prime time, you need a convenient way to copy all the files to your server. The vast majority of commercial web hosts offer support for something called FTP (File Transfer Protocol), which lets you easily copy a number of files at once (for details, see page 72).
- **Programming support.** Today, most web hosts support at least some of the dozens of different server-side programming languages available. When shopping for a web host, you'll come across their names—ASP, PHP, CGI, Perl, Python, Ruby, and so on. Although these languages are powerful, they usually require programmer credentials, and are beyond the scope of this book.

Note: Although this book doesn't cover server-side programs in depth, you'll learn about *client-side scripts* in Chapter 15. Client-side scripts run right inside website visitors' browsers, and are much more limited in ability than server-side programming platforms. Client-side scripts are commonly used for special effects like animated buttons. The nice thing about them is that even programming novices can drop a simple script into a web page and enjoy the benefits. But you don't need to worry about any of this right now, because unlike server-side programming, client-side scripts don't require any special support from your web hosting company.

- **Frills.** In an effort to woo you to their side, web hosts often pack a slew of frills into their plans. For example, sometimes they'll boast about their amazing, quick-and-easy website creation tools. Translation: They'll let you use a clumsy piece of software on their website to build yours. You'll end up with a cookie-cutter site and not much opportunity to express yourself. Steer clear of

these pointless features. More usefully, a web hosting company can provide site *statistics*—detailed information about how many visitors flock to your site daily or monthly. (In Chapter 11, you’ll find out about a free visitor tracking tool that runs circles around what most hosts provide.)

UP TO SPEED

The Riddle of Bandwidth

Most web hosting companies set their pricing, at least in part, on your web space and bandwidth needs. This can be a problem, because the average website creator has no idea how to calculate these numbers—let alone come up with realistic estimates.

Fortunately, you can save a lot of time and effort by understanding one dirty little secret: For the average personal or small-business site, you don’t need much disk space or bandwidth. You can probably take the smallest amounts on offer from any web hosting company and live quite happily.

If you still insist on calculating bandwidth, here’s how it works. Suppose you create a relatively modest website with

50 web pages, a pile of small web-optimized graphics, and a few downloadable documents (say, PDF files or Word documents). Altogether, this website occupies 10 MB of space.

Suppose your site is doing well, and receives about 30 visitors in a day. If each visitor visits every page (in other words, downloads your *entire website*), your daily bandwidth will rise to 300 MB (30 visitors \times 10 MB). Your monthly bandwidth will weigh in at 9 GB (30 days \times 300 MB). This bandwidth requirement is far less than typical startup web hosting packages, which often include 100 GB per month or more.

A Web Host Checklist

- ✓ **Web space.** 50 MB is acceptable if you’re getting free web space from your ISP. If you’re paying a monthly fee for web space, you’ll almost certainly get several gigabytes or more. This is probably far more than you’ll ever be able to use, even if you want to include photos, audio, or other large files. In fact, many web hosts offer *hundreds* of gigabytes of space, knowing that almost no one has the time to upload that much information.
- ✓ **Bandwidth.** You don’t need much. In fact, most people can ignore this number altogether. The only exceptions are if your website is absurdly popular; if you want to store extremely large files and let visitors download them; or if you’re showcasing a huge catalog of digital photos, music, or video, and you don’t want to use a third-party service like Flickr or YouTube. If you are in the small minority of people who needs huge amounts of bandwidth, look for a web host that promises unlimited bandwidth. That way, you don’t need to worry about exceeding your limit.
- ✓ **Domain name.** This is your identity—*www.You.com*. Ideally, your web host will throw in the domain name for free.
- ✓ **Email addresses.** These go with the domain name. Look for a web host that gives you five or more email addresses (sometimes called email *aliases*), because this lets

you use separate email addresses for different people or use them for different purposes. Also look for web-based access to your email so you can read it from any web-connected computer.

- ✓ **FTP access.** This lets you easily upload your files and is standard with most web hosting companies.
- ✓ **Tech support.** The best companies provide 24-hour tech support, ideally through a toll-free number or a live chat feature that lets you ask questions using your browser.
- ✓ **Server-side scripts.** Although web server programming is too complex for most ordinary people, this feature gives you some room to grow. If your website supports a server-side programming technology (like CGI, ASP, or ASP.NET), you could conceivably use someone else's script in your web pages to carry out an advanced task, like collecting visitor information with a form.
- ✓ **FrontPage extensions.** If you create your site using Expression Web (Microsoft's successor to FrontPage), you can use FrontPage server extensions to get a few extra frills. For example, you can count the number of visitors to a certain page and let visitors upload files to the server.
- ✓ **Coupons for advertising your website.** Some web hosts throw in a voucher for paid services, like a \$25 credit for Google AdWords. As you'll learn in Chapter 12, you can use AdWords ads to attract the right people to your site.

Choosing Your Host

With your requirements in mind, you're ready to start shopping for a web host. The following sections take you through your options.

Your ISP (Internet Service Provider)

As you may have already realized, your ISP—the company that provides your connection to the Internet—may have its own hosting services. In fact, ISPs sometimes include these services in your subscription price, meaning you may already have a dedicated amount of web space that you don't even know about. If you're in this situation, congratulations: You don't need to take any extra steps. If you're unsure, a quick call to your ISP will fill you in. Make sure you ask for “personal web space,” as many ISPs also provide large-scale web hosting packages for a monthly fee. The space is always far smaller than what you receive from a web hosting company, and it almost never includes a personal domain name (although you can purchase one separately).

Note: In many cases, your ISP may provide web hosting that you decide not to use. For example, they may not give you enough space, or they may force you to use their limited website creation software (which is a definite drag). For these reasons, you're usually better off paying a separate web hosting company.

Web hosting companies

Technically, any company that provides web space is a web host, but there's a class of companies that specialize in web hosting and don't do anything else. You can find these companies all over the Internet or in computer magazines. The disadvantage is that hosting companies always charge by the month. You won't get anything for free.

The sad truth is that researching web hosting companies online is almost impossible, because the Web is swamped with more advertisements for hosting than for cut-rate pharmaceuticals. Fortunately, there are many good choices. Table 3-1 lists just a few to get you started. If you're curious, be sure to check out these sites and start comparison shopping.

Table 3-1. A few of the Internet's many web hosting firms.

Name	URL
Brinkster	www.brinkster.com
DreamHost	www.dreamhost.com
GoDaddy	www.godaddy.com
HostGo	www.hostgo.com
Insider Hosting	www.insiderhosting.com
OCS Solutions	www.ocssolutions.com
Pair Networks	www.pair.com
Sonic.net	www.sonic.net

Don't expect an easy time finding honest web host reviews on the Web. Most websites that claim to review and rank web hosts are simply advertising a few companies that pay for a recommendation. Popular tech sites haven't reviewed web hosts in years, because a thorough analysis of even a fraction of them would require a massive amount of manpower. Their old reviews aren't much help either, because the quality of a hosting company can change quickly.

However, the Web isn't completely useless in your hunt. You can get information about hosting companies from a web discussion board where people like you chat with more experienced web hosts. One of the best is WebHostingTalk, which you'll find at <http://tinyurl.com/5zffwp>. The WebHostingTalk discussion board is particularly useful if you've narrowed your options down to just a few companies, and you'd like to ask a question or hear about other people's experiences.

As you consider different hosts, you need to sort through a dizzying array of options on different websites. In the next two sections, you'll practice digging through marketing haze to find the important information on the websites of two example hosting companies.

UP TO SPEED

Taming Long URLs with TinyURL

Keen eyes will notice that the URL mentioned on the previous page, the one that leads you to the WebHostingTalk forum, starts with the seemingly unrelated domain name TinyURL.com (<http://tinyurl.com>). WebHostingTalk forum's URL was deliberately been shortened using TinyURL, a website that provides a free URL redirection service. TinyURL is a handy tool you can use whenever you come across a URL that's so impractically long or convoluted that you'd ordinarily have no chance of jotting it down, typing it in, or shouting it over the phone.

Here's how to use TinyURL. First, copy your awkwardly long URL. Then, head to the website <http://tinyurl.com>, type or paste the URL into the text box on the front page, and click Make TinyURL. The site rewards you with a much shorter URL that starts with <http://tinyurl.com>. In the previous example, TinyURL changed the ridiculously

convoluted URL <http://www.webhostingtalk.com/forumdisplay.php?s=aa8768ada1bc5ddbd96e0578584cffce&f=1> to <http://tinyurl.com/5zffwp>.

Although this new URL doesn't mean anything, it's a heck of a lot shorter. Best of all, the tiny URL works just as well as the original one—type it into any browser and you'll get to the WebHostingTalk forum.

So how does this system work? When you type a tiny URL, your browser takes you to the TinyURL website. TinyURL keeps a long list of all the whacked-out URLs people provide as well as the new, shorter URLs it issues in their places. When you request a page, the site looks up the tiny URL in that list, finds the original URL, and redirects your browser to the site you really want. Here's the neat part: The whole process unfolds so quickly that you'd have no idea it was taking place if you hadn't read this box.

A web host walkthrough (#1)

Figure 3-8 shows how you can assess the home page for the popular web hosting company Aplus.Net. The company offers dedicated servers, standard hosting, domain name registration, and web design services. All four options are designed to help you get online, but the one for web hosting is what you're really looking for. The dedicated server option is a premium form of web hosting. It means that your website runs on its own server, a separate computer that doesn't host anyone else's site. This is primarily of interest to large business customers with high-powered sites that chew up computer resources. Most personal and small-business sites run on shared servers without any noticeable slowdown.

The top of the Aplus web page includes several tabbed buttons. The Domains tab gives you the option to transfer an existing domain name or park a domain for future use. The Hosting tab lets you see Aplus's hosting plans (see Figure 3-9), which is what you really want. The Web Design tab is mainly of interest to HTML-phobes. It lets you pay a web design team to craft all the HTML pages and graphics for your website. But where's the fun in that?

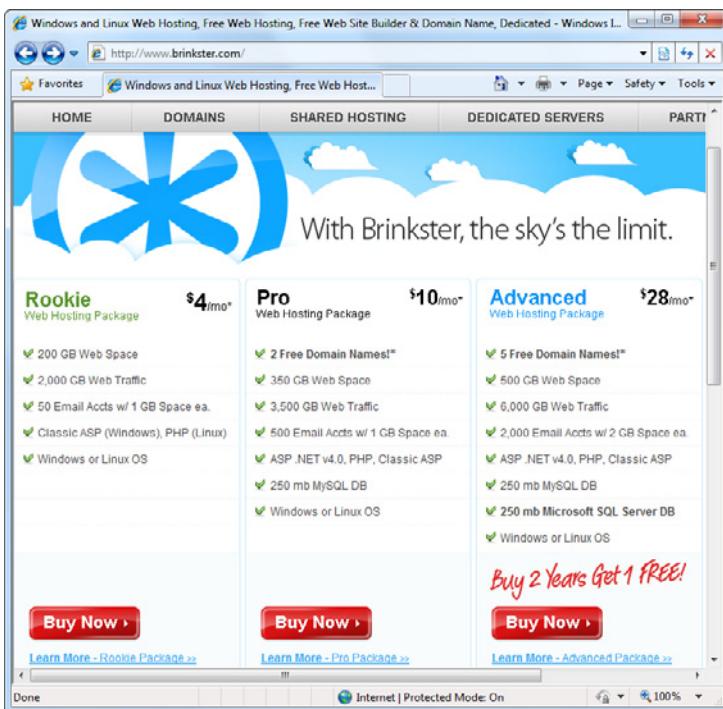
Figure 3-8: This page packs a lot of information. Click the Hosting tab to find out about Aplus's hosting plans (Figure 3-9). At the top of the page, Aplus lists a toll-free number for sales or support. Click the Chat Now button on the right edge of the page, and a chat window opens. Type your question here, and an Aplus technician gives you an immediate answer. If you're serious about signing up with Aplus, give both these options a try so you can evaluate its technical support.

Figure 3-9: At the end of your search, you discover that the cheapest option is \$6 per month for 15 GB of storage space for your website, and 250 GB of bandwidth. Aplus throws in a free domain name and 50 email addresses for good measure, along with FTP support. Click Learn More to get more technical details.

A web host walkthrough (#2)

Overall, the Aplus.Net search turned up a solid offer at a fair price, however, discerning web shoppers may be hoping to save a few dollars or get a little more space. Figure 3-10 profiles another hosting company—Brinkster. Brinkster's target audience includes personal site creators, small businesses, and developers, rather than large institutional customers. As a result, you just may find a better deal for your website.

Figure 3-10:
The Brinkster home page gets straight to the point with three basic packages. The Rookie package is a good bet for new web builders. At \$4 a month, it has space for a gargantuan 200 GB website, a huge helping of bandwidth (2000 GB), plus a free domain name and email addresses, too. Scroll over to the bottom-right corner (not shown here) for a live chat link.



Brinkster adds another wrinkle—not only can you choose the type of plan, but you can also pick the type of operating system used on the server where your website lives. Unless you're a programmer planning to create a web application that runs on a server, there's no reason to care about the web server's operating system. Assuming the hosting company does its job and distributes the websites they host over multiple computers, your site will be just as fast and reliable on any operating system. Think about it this way: When was the last time you asked yourself what operating system runs eBay (Windows) or Amazon (Linux)?

If you look carefully, you'll also find the Dedicated Server tab near the top of the Brinkster page. This super-premium option gives you an entire server to yourself. Big companies use this type of hosting to get rock-solid stability, but few people want to pay the extra cost, which can run well over \$100 a month.

After a tour of two hosting company's sites, you're ready to evaluate some on your own. Or, if you're really impatient, you can set up your site using one of the hosting companies you've seen. It doesn't take anything more than a couple of mouse clicks, and you'll be online in only a few hours.

Note: If your web host is letting you down, don't panic. It's not too hard to switch hosts. The key thing to remember is when you change hosts, you're essentially abandoning one server and setting up shop on another. It's up to you to copy your web pages to the new server—no one will do it for you. As long as you have a copy of your website on your personal computer (which you always should), this part is easy. If you're still a little skeptical of the company you choose, look for a 30-day, money-back guarantee.

Free Web Hosts

Not yet swayed by any of the hundreds of web hosting companies on the Web? Not tempted by the offer of a little web space from your ISP? If you're determined to save a monthly fee, there is a solution, but it may not be worth the aggravation.

The Web has a significant number of free web hosts. Free hosts are companies that give you a small parcel of web space without charging you anything. Sometimes it's because they hope to get you to upgrade to a cost-based service if you outgrow the strict limitations of the free package. Other times, they may just be interested in advertising revenue. That's because some free hosts force you to include an obnoxious ad banner at the top of your web pages.

Before you sign up for a free host, familiarize yourself with some of the headaches you can face:

- **Ad banners.** The worst free hosts force you to display their advertisements on your pages. If you'd like to crowd out your content with obnoxious credit-card commercials, this is the way to go. Otherwise, move on to somewhere new. It's finally possible to find free hosts that don't impose the Curse of the Blinking Banner Ad, so don't settle for one that does.
- **Unreliability.** Free web hosts may experience more down time, which means your website may periodically disappear from the Web. Or the web servers the host uses may be bogged down by poor maintenance or other people's websites, causing your site to slow to a crawl.
- **Unpredictability.** Free hosts aren't the most stable companies. It's not unheard of for a host to go out of business, taking your site with it and forcing you to look for a new web home in a hurry. Similarly, free hosts can change their requirements overnight, sometimes shifting from an ad-free web haven to a blinking banner extravaganza without warning.
- **Usage limits.** Some free web hosts force you to agree to a policy that limits the type of content you can put on your site. For example, you may be forbidden from running a business, selling ad space, or uploading certain types of files (like music, movies, or large downloads).

- **Limited tech support.** Many professional website operators say that what makes a good host isn't a huge expanse of free space or a ginormous bandwidth limit—it's the ability to get another human being on the phone at any hour to solve unexpected problems. Free web hosts can't afford to hire a platoon of techies for customer service, so you'll be forced to wait for help—if you get it at all.
- **Awkward uploads.** Many free hosts lack support for easy FTP uploading (see next section). Without this convenience, you'll be forced to use a time-consuming upload page.

Despite all these possible problems, many thrifty wallet-watchers swear by their free hosts (and the \$0 per month price tag). If you have the time to experiment and your business doesn't need rock-solid reliability and an immediate web presence, you might want to try out a few. Check out www.free-webhosts.com for a huge catalog of free hosts, which painstakingly details the space they give you and the conditions they impose. You'll also find thousands of user reviews. However, keep in mind that unscrupulous web hosts may pad the rankings with their own reviews, and any free host can abruptly change its offerings.

Transferring Files

Once you sign up for web hosting, you're ready to transfer some files to your web space. To perform this test, you can use Lee Park's résumé from the previous chapter (which you can download from the Missing CD page at www.missingmanuals.com/cds/caw3). The final version has the file name *resume5.htm*.

FTP

Ideally, your web host will provide FTP access. Using FTP, you can transfer groups of files from your computer to the server (or vice versa) in much the same way that you copy files from one folder to another in Windows Explorer or the Mac's Finder.

Before you can upload files using FTP, you need the address of the FTP server, as well as a user name and password. The latter two usually match your hosting account's user name and password but not always.

To upload files using FTP, you can use a standalone FTP program. However, in these modern times you probably don't need to. Windows includes its own built-in FTP browser that handles the task comfortably. Here's how it works:

1. **Open Windows Explorer.**

Right-click the Start button, and then choose Explore.

2. **Type the FTP address into the Windows Explorer address bar. Make sure the URL starts with *ftp://*.**

In other words, if you want to visit *ftp.myhost.com*, enter the URL *ftp://ftp.myhost.com*, not *http://ftp.myhost.com*, which incorrectly sends your computer off looking for web pages.

3. Enter your login information (see Figure 3-11).

Once you log in, you'll see your site's folders and files on the web server; you can copy, delete, rename, and move them in much the same way you do local folders and files. Since you haven't uploaded anything yet, the folder may be empty, or it may contain a generic *index.htm* file that displays an "under construction" message if someone happens to browse to the page.

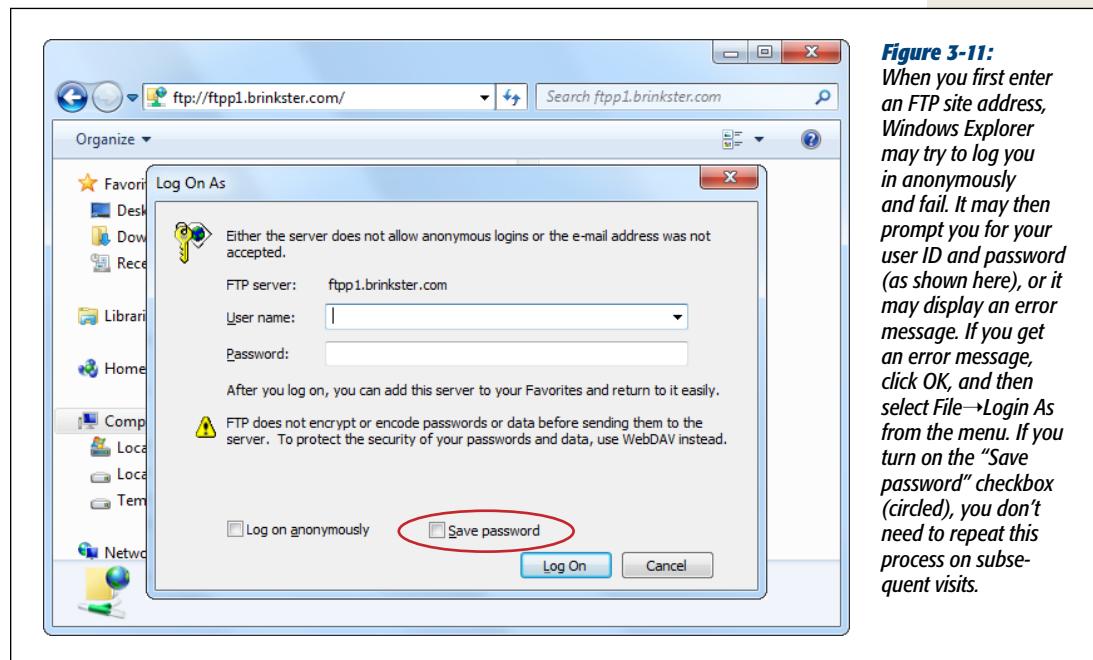


Figure 3-11:
When you first enter an FTP site address, Windows Explorer may try to log you in anonymously and fail. It may then prompt you for your user ID and password (as shown here), or it may display an error message. If you get an error message, click OK, and then select File→Login As from the menu. If you turn on the "Save password" checkbox (circled), you don't need to repeat this process on subsequent visits.

4. Copy your files to the server. The easiest way to do this is to drag the files from another open window, and then drop them in the FTP window.

Figure 3-12 shows the steps to upload the résumé example. Make sure you upload both the *resume5.htm* file and the linked picture, *leepark.jpg*, to the same folder on your site.

Tip: Drag-and-drop isn't the only way to transfer files. You can use all the familiar Windows shortcuts, including the Cut, Copy, and Paste commands in the Edit menu, and the Ctrl+C (copy) and Ctrl+V (paste) keyboard shortcuts.

If you're working on a Mac, you need to use a standalone FTP program. Fortunately, you have loads of free options, including the super-easy-to-use Rbrowser (available at www.rbrowser.com). Things work pretty much the same way they do for your Windows brethren. First, fire up Rbrowser. It'll ask you if you want to upgrade to the licensed version of the program; click FTP Only. On the next screen, fill in the

address of the FTP site; once you do, Rbrowser gives you the option to log in anonymously, so leave the Username and Password boxes blank. Then click Connect (Figure 3-13). Once that's out of the way, you can transfer your files by dragging them from a folder on your Mac to the Rbrowser window.

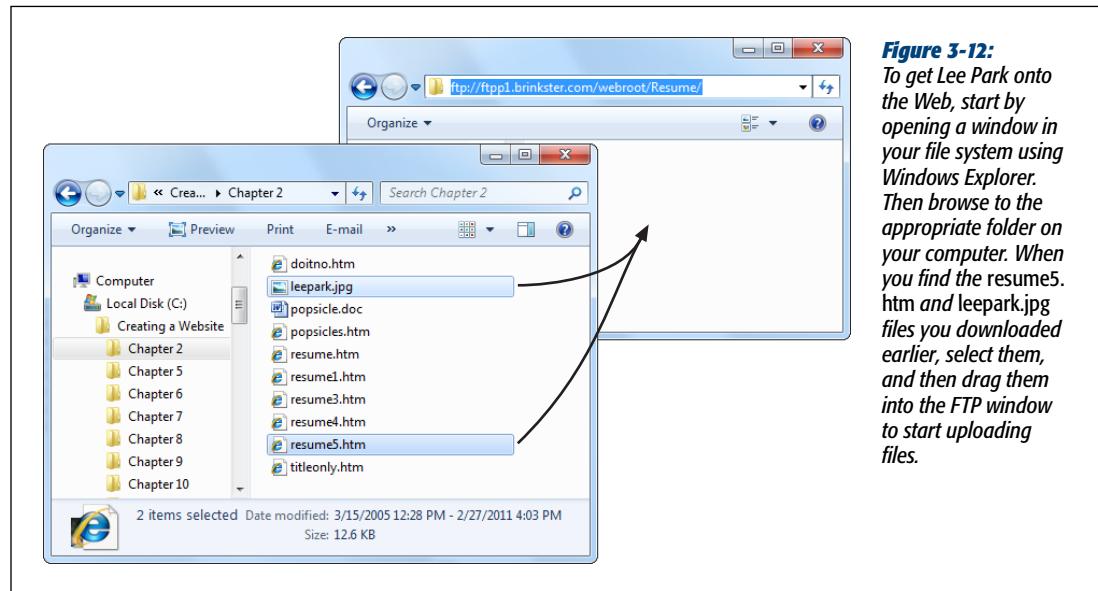


Figure 3-12:
To get Lee Park onto the Web, start by opening a window in your file system using Windows Explorer. Then browse to the appropriate folder on your computer. When you find the resume5.htm and leepark.jpg files you downloaded earlier, select them, and then drag them into the FTP window to start uploading files.

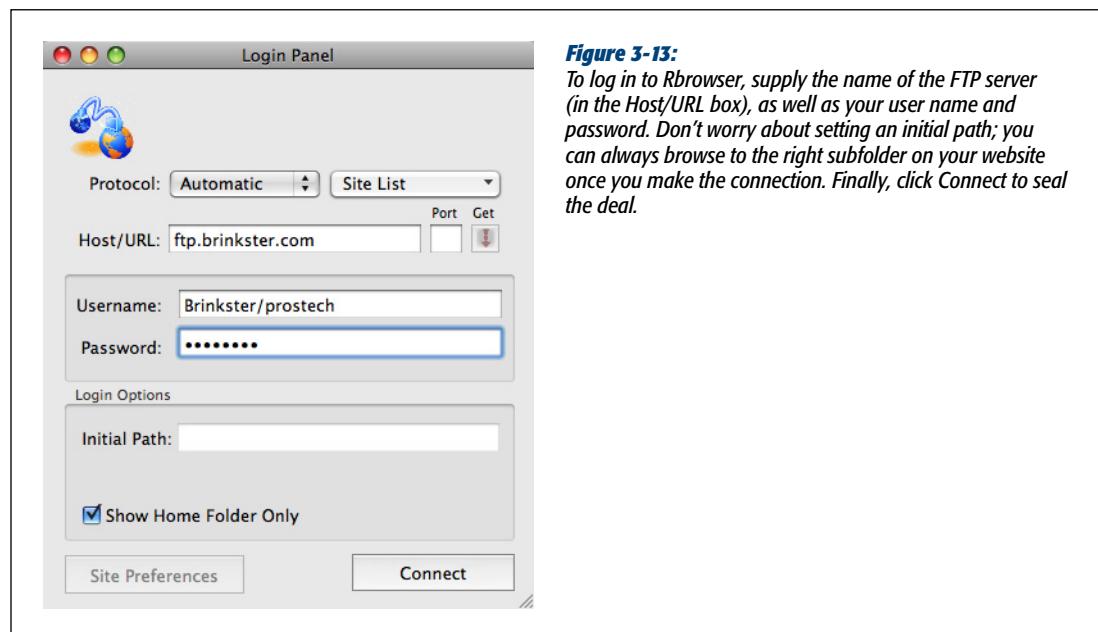


Figure 3-13:
To log in to Rbrowser, supply the name of the FTP server (in the Host/URL box), as well as your user name and password. Don't worry about setting an initial path; you can always browse to the right subfolder on your website once you make the connection. Finally, click Connect to seal the deal.

After you upload a file, you can test your work. Just enter your domain name followed by the web page name. For example, if you uploaded the résumé example to your website www.supersavvyworker.com, try requesting www.supersavvyworker.com/resume5.htm in your browser. You don't need to wait. Once you upload the file to your server, it's available almost instantly to any browser that requests it.

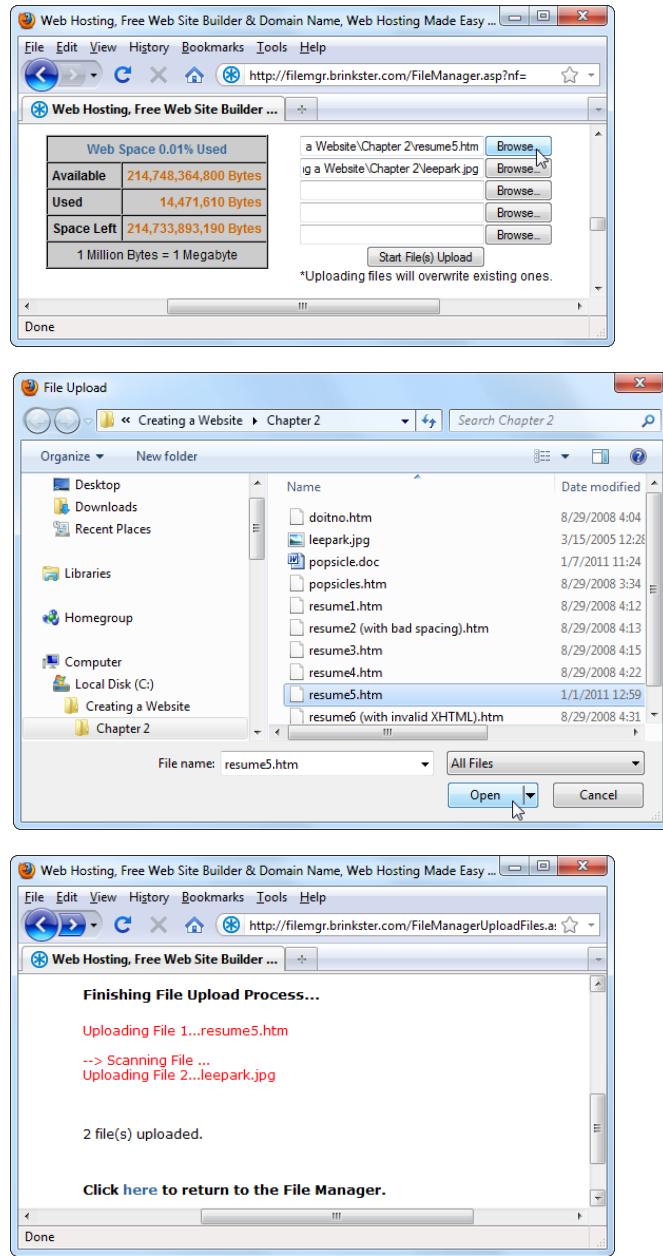
Browser-Based Uploading

Browser-based uploading is fairly easy, but it's not always convenient. The idea is that you go to a special web page on your host's site where you specify the files on your hard drive that you want to transfer to the server. Many hosting companies provide both browser-based and FTP-based uploading. If you're using a budget plan or a free web host, you may not have the FTP option at all. Instead, you'll upload files using your browser.

For browser-based uploading, visit your web hosting company's website and log in to your account with the user name and password you created when you signed up. Then browse through the icons until you find the file-management page.

Every web host has a slightly different version of the file-management page. Some pages mimic the look of a real FTP program, often using a Flash plug-in to create a slick interface that supports dragging and dropping batches of files. Others are more traditional, and force you pick your files one by one (Figure 3-14).

Unfortunately, the possibilities for mistakes with browser-based uploading are endless. The most common error occurs when you copy a large number of files at once. Not only is picking out each one time-consuming, the odds increase that you'll forget something. Other headaches include trying to upload files to different folders, or needing to rename or delete files after you upload them.

**Figure 3-14:**

Every web host's site looks a little different, but many feature a file-upload page that's a stack of text boxes. They all work the same way: First you click the Browse button (top), which brings you to an Open File dialog box (middle). Browse to the file you want, select it, and click Open. If you have several files to upload at once, repeat this process using different text boxes. When you've chosen all the files you want (or just run out of text boxes), click Start File Upload, and then wait until the files are copied and you get a confirmation message (bottom).

Power Tools

In Chapter 2, you built your first HTML page with nothing but a text editor and a lot of nerve—the same way all web-page whiz kids begin their careers. To really understand HTML (and to establish your HTML street cred), you need to start from scratch.

However, very few web authors stick with plain-text editors or use them to create anything other than simple test pages. The average HTML page is filled with tedious detail. Try to write every paragraph, line break, and formatting tag by hand, you'll probably make a mistake somewhere along the way. Even if you don't, it's hard to visualize a finished page when you spend all day staring at angle brackets. This is especially true when you tackle more complex pages, like those that introduce graphics or use multicolumn layouts.

The downside to outgrowing Notepad or TextEdit is the expense. Professional web design tools can cost hundreds of dollars. At one point, software companies planned to include basic web editors as part of operating systems like Windows and Mac OS. In fact, some older versions of Windows shipped with a scaled-down web editor called FrontPage Express. (Most Macs include a severely truncated editor called iWeb, which limits you to ready-made templates and doesn't let you touch a line of HTML.) But if you want a full-featured web page editor—one that catches your errors, helps you remember important HTML elements, and lets you manage your entire site—you have to find one on your own. Fortunately, there are free alternatives for even the most cash-strapped web weaver.

In this chapter, you'll learn how web page editors work and how to evaluate them to find the one that's right for you. You'll also tour some of the better free and shareware offerings out there. Most web page editors are surprisingly similar, so this chapter

helps you get started with your tool of choice, whether that's Adobe Dreamweaver, Microsoft Expression Web, or a nifty piece of freeware.

Choosing Your Tools

Tools like Notepad and TextEdit aren't all that bad for starting out. They keep page development simple and don't mess with your HTML (as a word-processing program would). Seeing the result of your work is just a browser refresh away. So why are you destined to outgrow your favorite text editor? For a number of reasons, including:

- **Nobody's perfect.** With a text editor, it's just a matter of time before you make a mistake, like typing `` instead of ``. Unfortunately, you might not realize your mistake even when you view your page in a browser. Browsers do their best to compensate for HTML inaccuracies, even if it means obscuring the real problems in your page. A good web page editor can highlight faulty HTML and help you correct it.
- **Edit-Save-Refresh. Repeat 1,000 times.** Text editors are convenient for small pages. But what if you're trying to size a picture perfectly or line up a table column? You need to jump back and forth between your text editor and your web browser, saving and refreshing your page each time, a process that can literally take hours. With a good web page editor, you get conveniences like drag-and-drop editing to fine-tune your pages—make a few adjustments, and your editor tweaks your HTML appropriately. Editors also have a preview mode that lets you immediately see the effect of your HTML edits on a web page, with no browser required.
- **Help, I'm drowning in HTML!** One of the nicest little frills of a web page editor is color-coded HTML. Color-coding makes those pesky tags stand out against a sea of text. Without this feature, you'd be cross-eyed in hours.
- **Just type ``.** To create a bulleted list, of course. You haven't forgotten already, have you? The truth is, most web authors don't memorize every HTML element there is. With a web editor, you don't need to. If you forget something, there's usually a help link or a menu command to compensate. Without a tool to guide you, you're on your own.

Of course, using a graphical web page editor has its own risks. That's why you started out with a simple text editor and why you'll spend a good portion of this book learning more about HTML (and a few related standards). If you don't understand HTML properly, you can fall into a number of traps.

For example, you might use a slick page editor to apply fancy fonts to your text. Imagine your surprise when you look at your work on a computer that doesn't have those fonts installed; your page reverts to an ugly or illegible typeface. (Chapter 6 has more about this problem.) Similarly, your editor can unwittingly lead you to insert

elements that trip up some browsers or graphics that won't display properly on other computers. Finally, even with the best editor, you'll spend a significant amount of time looking at raw HTML to see exactly what's going on, clean up a mess, or copy and paste useful bits to other pages.

Types of Web Page Editors

Web page editors come in many flavors, but they all tend to fall into one of three categories:

- **Text-based** editors require you to work with the text and tags of raw HTML. The difference between an ordinary text editor (like Notepad) and a text-based HTML editor is convenience. Unlike Notepad or TextEdit, text-based HTML editors usually include buttons that let you quickly insert common HTML elements or element combinations, and a one-click way to save your file and open it in a separate browser window. Essentially, text-based HTML editors are text editors with some useful web-editing features stapled on top. Website creators prefer HTML editors that have as many of these features as possible, but still provide a text-only option.
- **Split window** editors also make you write HTML by hand. The difference is that a separate window shows the results of your work *as you type*. In other words, you get a live preview, so you don't need to switch back and forth between text editor and browser to see what you've accomplished.
- **WYSIWYG** (What You See Is What You Get) editors work more like word processors. Instead of writing HTML tags, you type in a page's text, format it, and insert pictures just as you would in a word-processing program. Behind the scenes, the WYSIWYG editor generates your HTML markup.

Any of these editors makes a good replacement for a simple text editor. The type you choose depends mainly on how many features you want, how you prefer to work, and how much money you're willing to shell out. The best web-editing programs blur the lines between these different types of editor, giving you the freedom to switch back and forth between HTML and WYSIWYG views.

No matter which type of editor you use, you still need to know a fair bit about HTML to get the results you want. Even if you have a WYSIWYG editor, you'll almost always want to fine-tune your markup by hand. Understanding HTML's quirks lets you determine what you can and can't do—and the strategies you need to follow to get the most sophisticated results. Even in a WYSIWYG editor, you'll inevitably look at the HTML underbelly of your web pages.

FREQUENTLY ASKED QUESTION

Save As HTML

My word-processing/page layout/spreadsheet program has a feature for saving documents as web pages. Should I use it?

Over the last decade, the Internet has become the hottest marketing buzzword around. Every computer program imaginable is desperate to boast about new web features. For example, virtually every modern word processor has a feature for exporting your documents to HTML. Don't use it.

HTML export features don't work very well. Often, the program tries to wedge a document designed for one medium (usually print) into another (the Web). But word processor documents just don't look like web pages. They tend to have larger margins, fancier fonts, more text, more generous spacing around that text, no links, and a radically different layout. When you export a document, your word

processor tries to preserve these details, in the process creating pages that aren't readable or attractive when viewed in a web browser.

Another problem is the fact that HTML export features often create wildly complex markup. You end up with an ungainly web page that's nearly impossible to edit because it's choked with formatting details. (Dreamweaver even has a tool that aims to help you with the cleanup; look in the menu under Commands→Clean Up Word HTML.) And if you want to convert one of these pages into stricter, cleaner HTML, you need to do it by hand.

The lesson? If you can, steer clear of the "Save as HTML" command. You're better off copying and pasting the contents of your document into an HTML file as plain text, and then formatting it with HTML tags on your own.

Finding a Free Web Page Editor

Unless you're one of the lucky few who already has a copy of a cutting-edge web page editor like Dreamweaver or Expression Web, you're probably wondering how you can find a good piece of software for as little money as possible. After all, the Web is all about getting goodies for free. While you can't find an industrial-strength Dreamweaver clone for free, you *can* get a good basic editor without opening your wallet. Here's how:

- If you like to do your own research (always a good idea) and you don't mind installing several dozen programs on your computer until you find what you like, head to a top shareware site like www.download.com. To dig up some contenders, search for "HTML editor."

Note: Shareware is software that's free to try, play with, and pass along to friends. If you like it, you're politely asked to pay for it (or not-so-politely locked out when the trial period ends). Freeware is software that has no cost at all—if you like it, it's yours! Usually, you won't get niceties like technical support. Some freeware is supported by donations. To make sure your shareware is virus- and spyware-free, download it from a reputable source like www.download.com.

- You can also browse a comprehensive list of web page editors on Wikipedia at http://en.wikipedia.org/wiki/Comparison_of_HTML_editors. This list is noteworthy for the surprisingly thorough information it provides. For example, it details the price of each editor and the operating systems each works with. Most importantly, it points out which editors you can get for free.

You'll quickly find out that there's a sea of free editors out there. Many have awkward and clunky button and menu arrangements. Some have outright errors. Finding one that's right for you might take a little time.

The following sections describe four worthwhile editors that don't cost a penny.

BlueGriffon

BlueGriffon is a nifty, lightweight editor that's supremely easy to use. Best of all, BlueGriffon comes in a version for all three major operating systems: Windows, Mac OS X, and Linux.

BlueGriffon uses the same rendering engine that powers the Firefox browser. It's an "open source" project, which means that you can not only download and copy it for free, but you can, if you're a programmer type, browse through the source code and submit improvements. As a web-head, you're most likely to fall in love with BlueGriffon's multiple views, which let you make changes in either a text-only Source view or a slick WYSIWYG view that previews your pages (see Figure 4-1).

BlueGriffon has a somewhat tangled history. Many of the same developers worked on its predecessor, the popular free editor Nvu, until 2006. When development stopped on Nvu, *another* team of coders used Nvu as the basis for a patched-up web editor called KompoZer. But now that BlueGriffon is on the scene, it trumps both of these other editors.

BlueGriffon's key drawback is that it's new and evolving rapidly. On the positive side, you'll get good support for the new features in HTML5. (BlueGriffon's website provides regular "beta builds" of newer but untested versions of the editor that pack in the latest new features; look for them under the site's Fresh Meat link.) The definite downside is you should expect bugs and minor glitches. At the time of this writing, the current version of BlueGriffon is 0.8, which means the development team hasn't quite reached the stability and reliability standards of a professional 1.0 release.

To download BlueGriffon or read about its progress, go to www.bluegriffon.org.

Tip: Like many more advanced (and more expensive) web editors, BlueGriffon includes an integrated spell-checker. If BlueGriffon finds an unrecognized word (like "hizzle" in Figure 4-1), it underlines the word in red. You can then right-click that word to pick from a list of suggested corrections.

**Figure 4-1:**

Top: Instead of making you work with raw HTML, BlueGriffon's WYSIWYG view lets you format text just as you would in a word processor. To switch from one view to another, use the tabs at the bottom of the window (circled).

Bottom: To fine-tune your HTML markup, switch to the Source view. BlueGriffon gives you handy line numbers for reference and color-codes your HTML elements.

Amaya

Amaya is a web page editor with real history—It's been around for more than a decade. It helps that the World Wide Web Consortium, the heavyweight international standards organization that helps standardize the Web, has adopted Amaya. Like BlueGriffon, Amaya supports the big three operating systems, which is a rare asset—most free editors, including the two you'll learn about next, are for Windows PCs only. Amaya's options for viewing web pages do BlueGriffon one better (see Figure 4-2).

Figure 4-2:

Top: Open a web page in Amaya, and you start in Design view.

Bottom: Select Views→Show Source from the menu bar, and Amaya displays this split window view, which lets you jump back and forth between Design view and your HTML markup. Best of all, you can make changes in either window. If you have a widescreen monitor, you don't need to stick with top and bottom stacking; choose Views→Split View Vertically to put the two views side by side.

Amaya isn't as polished as a professional page editor, and finding the command you want in its cluttered menu and toolbars is often a chore. Its lack of recent updates is also disappointing, especially for HTML5 fans, who will have to type in new tags on their own. Still, for a free editor, Amaya is a barn burner of a bargain. To give Amaya a whirl, go to www.w3.org/Amaya.

HTML-Kit

HTML-Kit began its life as a slightly eccentric Windows web page editor with a screen layout that only Bill Gates's mother could love. That version, now called HTML-Kit 292, still exists. It's free, relatively reliable, and ridiculously customizable.

But the same company now offers a revamped version called, confusingly enough, HTML Kit Tools (\$59). The new version has a slicker interface and better support for HTML5.

Unlike BlueGriffon and Amaya, HTML-Kit doesn't provide a WYSIWYG editing mode. Instead, it has a split-preview editor, which means you can see a live preview of your HTML document as you edit it. To download the free version (or a trial of the new, for-pay version), go to www.htmlkit.com.

CoffeeCup Free HTML Editor

CoffeeCup Free HTML Editor is a scaled-down version of the full-blown Windows product of the same name (minus the word "Free"). The full version (\$49) has both a text-only mode and a WYSIWYG mode, but the free version switches off the WYSIWYG mode. Get your copy at www.coffeecup.com/free-editor.

Professional HTML Editors

Fed up with settling for a low-powered web page editor and an editing environment seemingly designed by M. C. Escher? If you're ready to move on to a professional web design package, take heart—your choice is surprisingly simple. You'll find really only two top-tier editors on the market today.

- **Adobe Dreamweaver** is the favorite of graphic designers and hard-core HTML experts. Packed with features, it gives you fine-grained control of every HTML ingredient. Because it's the market leader and the tool of choice for most professionals, however, it carries a hefty price tag: \$400 for the standard edition.
- **Microsoft Expression Web** is a complete rebuild of Microsoft's long-running FrontPage web editor. It offers many of the same high-powered features as Dreamweaver at less than half the cost (\$150 at the time of this writing). If you aren't out to impress a crowd of black-turtleneck-wearing web designers, it just might be the best value for your dollar.

Note: Confusingly, Microsoft Expression Web has a distant cousin with most of the same features. It's named, in true long-winded Microsoft style, Microsoft Office SharePoint Designer. Like Expression Web, SharePoint Designer is a successor to the now-obsolete FrontPage. Unlike Expression Web, it has some optional features that work with Microsoft SharePoint, a web-based program that big businesses use to share office documents and help employees collaborate. If you're buying a Microsoft web page editor on your own dime, you'll almost certainly stick with Expression Web. But if you already have access to SharePoint Designer (say, through the company you work for), feel free to substitute it instead.

One of the reasons these products are so much better than their competitors is that they include a lot of tools you're sure to need once you start designing web pages. For example, both let you create style sheets (an advanced feature you'll learn about in

Chapter 6), resize images and drag them around your pages, and manage an entire website. Expression Web even includes a tool for generating fancy buttons.

Another reason is that they're just so darned easy to use. Even though both come packed with sophisticated features, editing a simple HTML file couldn't be easier.

In the past, Dreamweaver had a reputation for being complicated enough to scare away HTML newbies. On the other hand, FrontPage (Expression Web's predecessor) was known for being easy to use but possessed of a few bad habits—like inserting unnecessary elements into your code or relying on frills that worked only if your web server supported the FrontPage server extensions. However, recent versions of both programs tackle these weaknesses. Now, Dreamweaver is virtually as easy to use as Expression Web, and Expression Web is almost as mature and well-rounded as Dreamweaver. In fact, common tasks in these two programs are surprisingly similar. The bottom line? You can't go wrong with either.

If you're still itching to be convinced, try a free 30-day trial of either product. Go to www.microsoft.com/Expression/try-it for a free trial version of Expression Web, or to www.adobe.com/products/dreamweaver to download a working Dreamweaver demo. And, for an in-depth exploration of every Dreamweaver feature, check out *Dreamweaver CS5: The Missing Manual*.

Tip: Looking for the best but trying to save some bank? Both Adobe and Microsoft offer sweetly discounted *upgrade pricing*. For Adobe, you need to own a previous version of Dreamweaver or GoLive to get the deal, but Microsoft will qualify you for having a retail version of *any* Adobe Creative Suite, Microsoft Expression, or Microsoft Office program. Alternatively, you may be able to find academic (or "student and teacher") editions of both programs at specialty retailers, like college campus computer stores. These editions are scaled down but still powerful, and they have a much lower price tag. You'll need to prove you're a student or starving academic to get in on the action.

Working with Your HTML Editor

Once you choose a web page editor, take it for a spin. Software companies have spent the last decade copying features from their competitors and as a result, common tasks in Expression Web, Dreamweaver, and many free web page editors are startlingly similar. No matter which program you use, the following sections will teach you the basics: how to create a sample HTML document and get it online, all without leaving the comfort of your editor. Once you're at home with your editor, you can move on to the rest of the book and learn more about how HTML works.

Note: Although future chapters won't lead you step by step through any of these editors, look for boxes and tips that point out occasional shortcuts, tricks, and techniques for your favorite one. The only exception is the keenly important page template feature that both Dreamweaver and Expression Web provide—you'll explore that in Chapter 10.

Starting Out

Your first step is to launch your web editor by double-clicking the appropriate desktop icon or making a quick trip to the Start menu (on a Windows computer) or Finder→Applications (for Mac OS).

Some editors, such as Amaya, start you off with a tip of the day. If so, close this window to get to the main program.

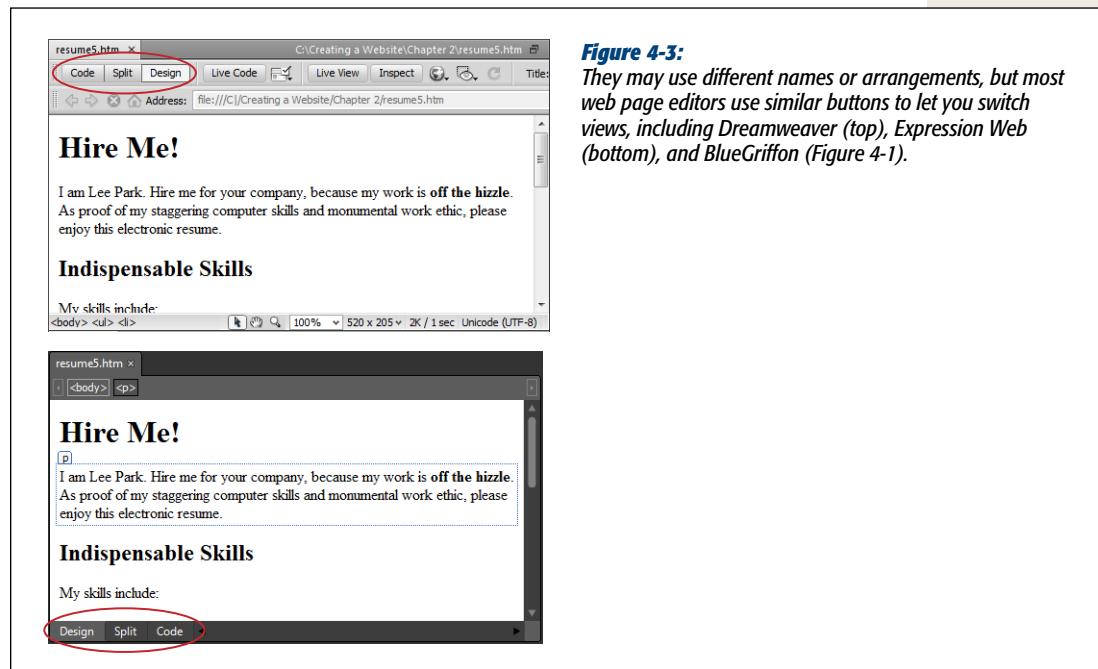
Once your editor starts up, you’re ready to open a file and start editing. Usually, that means using the File→Open command, just as in any other self-respecting program. To try out your editor, open one of the HTML file samples you worked on in Chapter 2 (also available on the Missing CD page at www.missingmanuals.com/cds/caw3). The examples in this chapter use the file *resume5.htm*.

Multiple Views

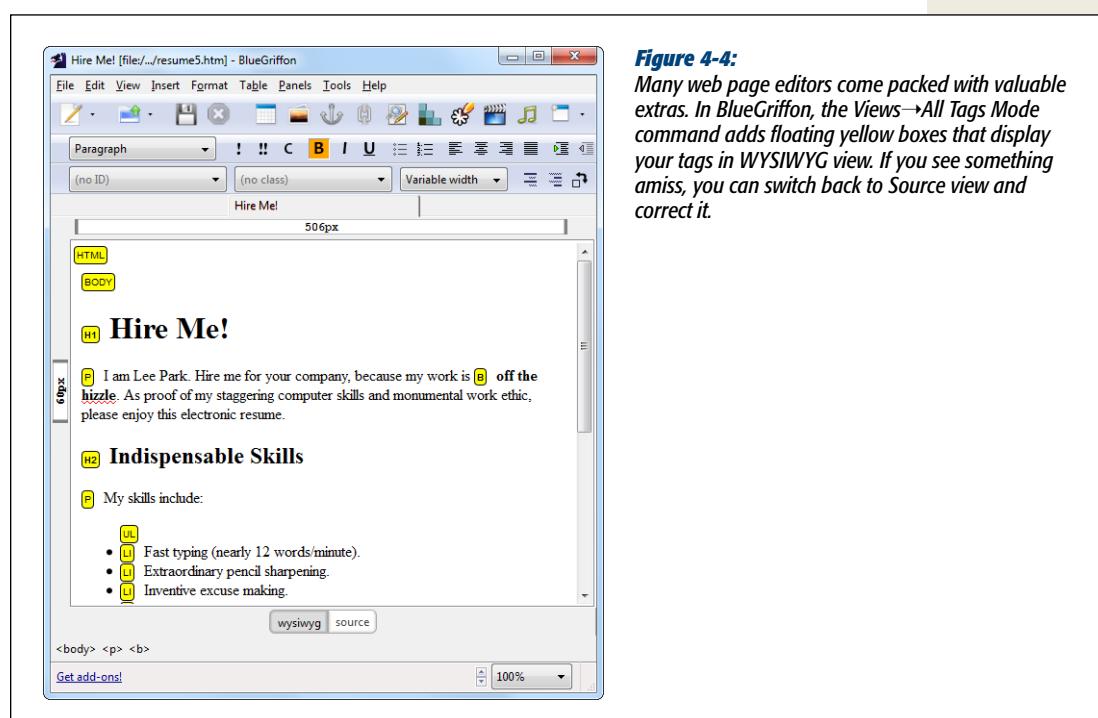
As you’ve already learned, you can look at an HTML document in several ways, depending on whether you want the convenience of a word processor-like layout or the complete control of working directly with HTML markup. Most web page editors give you a choice and let you switch rapidly from one to the other. To switch views, you need to find a small series of buttons, usually displayed just above or below the document you’re working on. Figure 4-3 helps you spot these buttons. One exception is Amaya. It lacks these handy buttons and forces you to travel to the Views menu to make the switch.

Most editors start you out in a WYSIWYG view that displays formatted HTML—in other words, an approximation of what a page will look like in a web browser. When you switch to the HTML code view, you see the real story—the familiar text-only display of color-coded tags and text. These views represent the two staples of HTML editing. However, the most useful choice just might be the split view, which shows both views at once. Most commonly, you’ll use this view to edit HTML tags and preview the results as you type. You can also work the other way, however, editing the WYSIWIG preview and seeing what HTML tags your editor inserts, which is a great way to learn HTML.

Some editors give you interesting hybrid views. For example, BlueGriffon has an HTML Tags feature that shows HTML tags in floating yellow boxes. To see it, switch to WYSIWYG view and choose View→All Tags Mode (Figure 4-4). Amaya offers two more specialized views. You can get an overview of the structure of your entire page using Views→Show Structure, and you can review all the links to other web pages (as explained in Chapter 8) using Views→Show Links.

**Figure 4-3:**

They may use different names or arrangements, but most web page editors use similar buttons to let you switch views, including Dreamweaver (top), Expression Web (bottom), and BlueGriffon (Figure 4-1).

**Figure 4-4:**

Many web page editors come packed with valuable extras. In BlueGriffon, the Views→All Tags Mode command adds floating yellow boxes that display your tags in WYSIWYG view. If you see something amiss, you can switch back to Source view and correct it.

Creating a Web Page in Code View

The best way to understand how your web page editor works is to create a new HTML document. Begin by working in Code view, and try to replicate one of the resumes you developed in Chapter 2 (the one on page 37 is ideal).

Open your program and follow these steps:

1. Start by choosing File→New to create a new web page.

Some web page editors start you out with a new, blank page, in which case you can skip the rest of these instructions. But most open some sort of New Document window that lets you choose the type of file you want to create.

2. Choose the type of page you want to create.

Many web editors give you a number of ready-made page designs to get you started. For example, in Dreamweaver and Expression Web, you can create pages with a variety of multicolumn layouts (more on that in Chapter 9). You’re best off to avoid these until you learn enough to create your own unique designs. For now, start with a blank, plain-vanilla HTML page. In Dreamweaver, choose Blank Page and pick HTML in the Page Type list. In Expression Web, choose the General group, and then pick HTML.

Many editors also let you choose a doctype (page 31) for your new web page. In Dreamweaver, you make your selection from the DocType list in the New Document window. In Amaya, you use the similar Document Profile list in the New XHTML Document window. In Expression Web, you need to take an extra step—first, click the Page Editor Options link in the New window, and then find the Document Type Declaration list in the Page Editor Options window. And in BlueGriffon, you need to choose File→New Wizard instead of File→New to get a doctype choice. But don’t worry if you can’t find the option you need in your web editor, because you can easily edit the doctype by hand after you create the page.

3. Create the document.

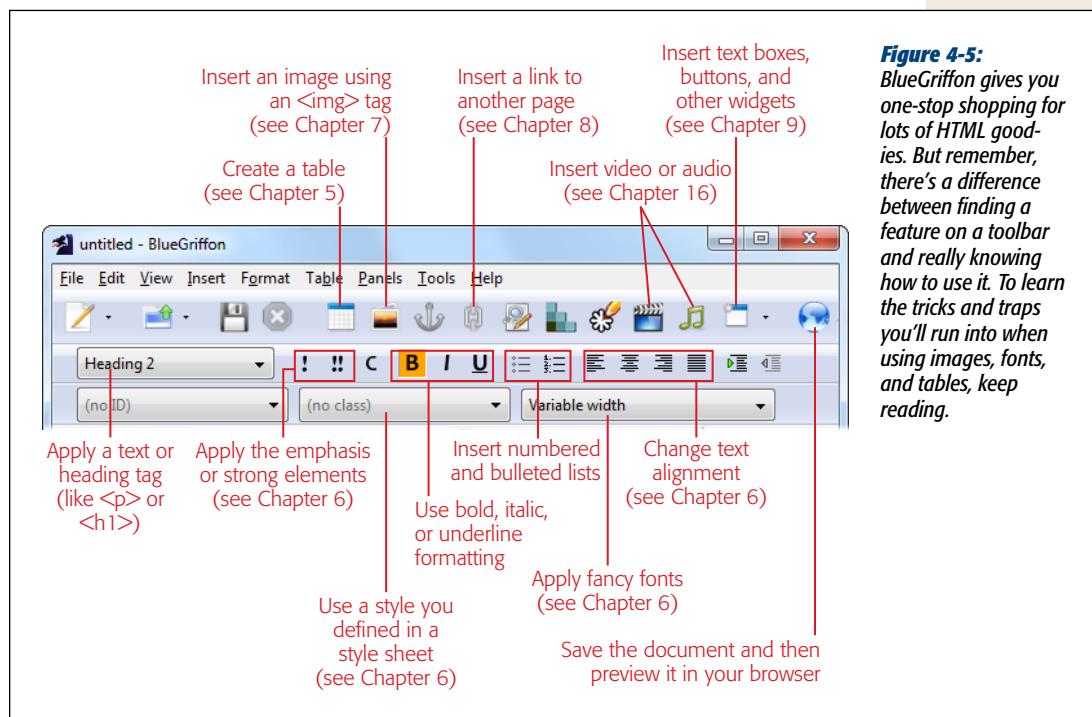
To make your selections official and actually create the document, click Create or OK (depending on the program). The editor displays your new blank document, which contains a bare HTML skeleton. In Code view, you see the basic `<html>`, `<head>`, and `<body>` elements.

Make sure you’re in Code view so you have complete control over the HTML markup. Enter all the tags and text for the résumé from beginning to end, just as though you were using Notepad or TextEdit. (You can pop back to Chapter 2 for a refresher on HTML code-writing basics.) Along the way, you notice a few shortcuts. For example, when you start to type a tag in Dreamweaver or Expression Web, a drop-down menu appears with suggestions. You can choose a valid HTML tag from the list or just keep typing. Also, when you add the start tag for a container element (like `<h1>` for a heading), many web editors automatically insert the end tag (like `</h1>`) so you won’t forget it.

Creating a Web Page in WYSIWYG View

Creating and formatting a page in WYSIWYG view is a more interesting challenge, because you need to know where to find the various formatting options in your editor. Dreamweaver and Expression Web help you out by packing a fair bit of HTML smarts right into their toolbars. To add an element in WYSIWYG view, you first select the piece of text you want to format, and then click the appropriate toolbar button. You can then switch to the HTML Code view to verify that you got the result you expected. For example, to make a piece of text bold, select it and then click the toolbar button marked B. Behind the scenes (in your actual HTML markup), the editor inserts a `` tag just before your selection and a `` tag just after it.

Figure 4-5 shows you the most useful toolbar buttons in BlueGriffon. But no matter which web page editor you use, you need to spend a fair bit of time wandering through its toolbars and menus to find the most convenient way to get things done.



Tip: It's up to you whether you create your pages in Code or WYSIWYG view. The WYSIWYG view is always quicker and more convenient at first, but it can leave you with a lot of HTML to check and review, adding to future complications.

To practice your WYSIWYG editing, re-create one of the example pages from Chapter 2 (the mini-résumé you see on page 40 works best here). Instead of entering the tags by hand, however, enter the text and format it using the toolbar and menu options in your editor.

Create a new document as described above, and then follow these steps:

1. Switch to WYSIWYG view, and then type the title “Hire Me!”
The text appears at regular size.
2. Select the text, and then find a toolbar or menu option that will convert it to a heading by adding the `<h1>` and `</h1>` tags.

To mark up the text, you need to get a bit more intimate with your web page editor and its menu and toolbar options.

In Expression Web (and BlueGriffon), the quickest approach to apply a level-1 heading is to select Heading 1 from the drop-down Style list. In Dreamweaver, you have two worthwhile choices. You can use the shortcut key combination (Ctrl+1) for instant gratification, or you can use the drop-down list in the Properties window (choose Window→Properties if you can’t see it). In Amaya, you need to click the T1 icon in the toolbar-like panel that appears on the right side of the window.

3. Press Enter to move to the next paragraph.

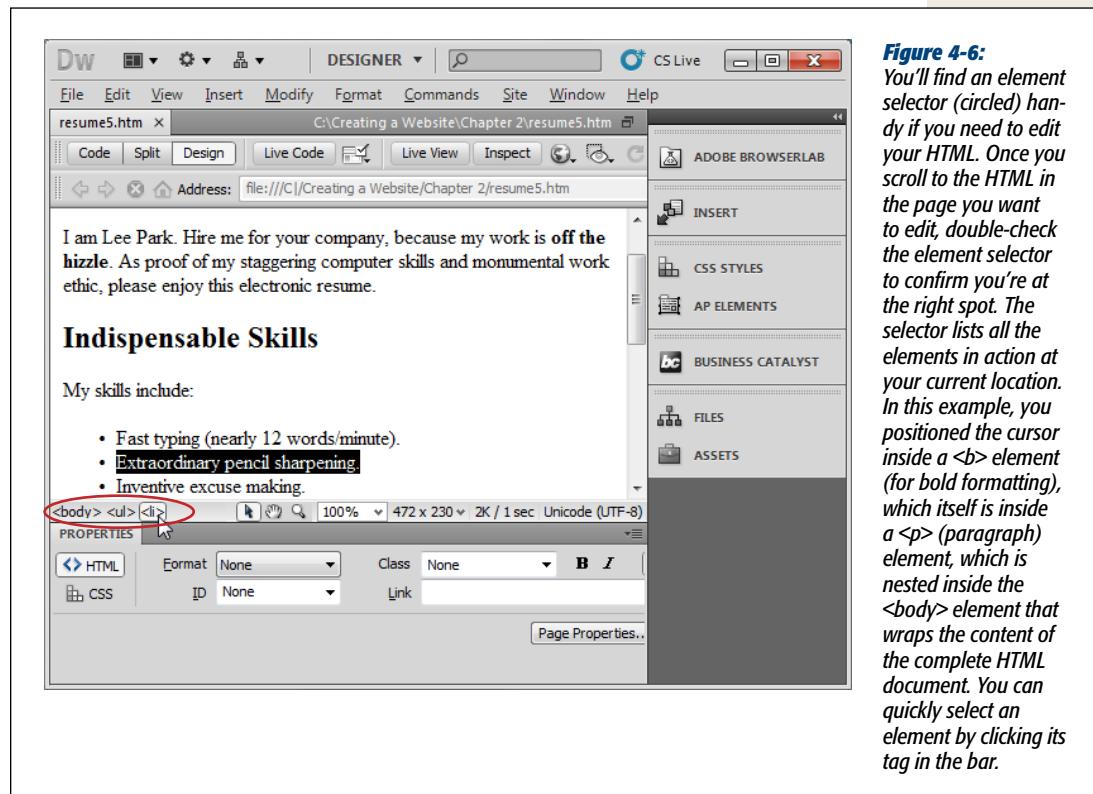
Your typeface reverts to normal, and you can begin typing the rest of the document. Your next challenge is creating the bulleted list. Dreamweaver, Expression Web, BlueGriffon, and Amaya all have a button for bulleted lists in the toolbar, if you can ferret it out.

It’s easy to lose yourself in a thicket of tags. To make orienting yourself easier, Expression Web, Dreamweaver, BlueGriffon, and Amaya all include a quick element selector bar at the top or bottom of your document (Figure 4-6).

4. For a change of pace, try inserting a picture.

BlueGriffon’s Image button lets you do so easily. Dreamweaver and Expression Web have similar buttons, but it’s easier to head straight to the menu (choose Insert→Image in Dreamweaver and Amaya, and Insert→Picture→From File in Expression Web).

Note: For this test, you should put the picture in the same directory as your web page. Otherwise, some editors may add an `` element that’s linked to a specific location on your hard drive. This is a problem: Web visitors can’t tap into your hard drive, and so they won’t see the picture. To double-check that everything’s in order, look at the `` element in HTML view, and make sure the `src` attribute doesn’t start with `file:///`. If it does, edit it by hand so that the `src` attribute has just the file name, like the `` element you used in Chapter 2 (page 40).

**Figure 4-6:**

You'll find an element selector (circled) handy if you need to edit your HTML. Once you scroll to the HTML in the page you want to edit, double-check the element selector to confirm you're at the right spot. The selector lists all the elements in action at your current location. In this example, you positioned the cursor inside a **** element (for bold formatting), which itself is inside a **<p>** (paragraph) element, which is nested inside the **<body>** element that wraps the content of the complete HTML document. You can quickly select an element by clicking its tag in the bar.

- When your editor prompts you to pick an image file, browse to the leepark.jpg sample, and then select it. (You can download this image from the Missing CD page at www.missingmanuals.com/cds/caw3.)

The program adds the appropriate `` element to your HTML code. Once you insert the picture, you'll really start to appreciate the benefits of the WYSIWYG view. In all the web page editors covered here, you can drag the picture to move it or drag the picture's borders to resize it.

Managing a Website

Most web page editors don't limit you to working on a single page at a time. Dreamweaver, Expression Web, BlueGriffon, and Amaya all let you open more than one document simultaneously and switch among them using tabs at the top of the document window (see Figure 4-7). This is particularly handy if you need to make the same change to a whole batch of pages, or if you want to cut a bit of content from one page and paste it into another.

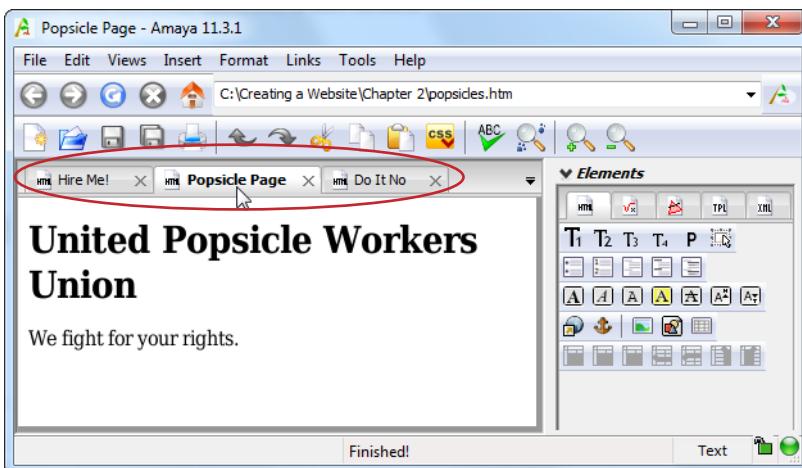


Figure 4-7:
With tabbed editing, your web page editor creates a new tab each time you open a file. You can switch from one page to another by clicking the tabs, which usually sit above the top of the page (and underneath the editor's menus and toolbars). Here, Amaya has three web pages open at once. To close a document, click the X on the right side of the tab.

Along with the ability to edit more than one web page at once, many editors also let you manage your entire website. The following sections describe the process for Dreamweaver and Expression Web. In both cases, your first step is to define your site's folders and files so your editor knows just which documents make up your site.

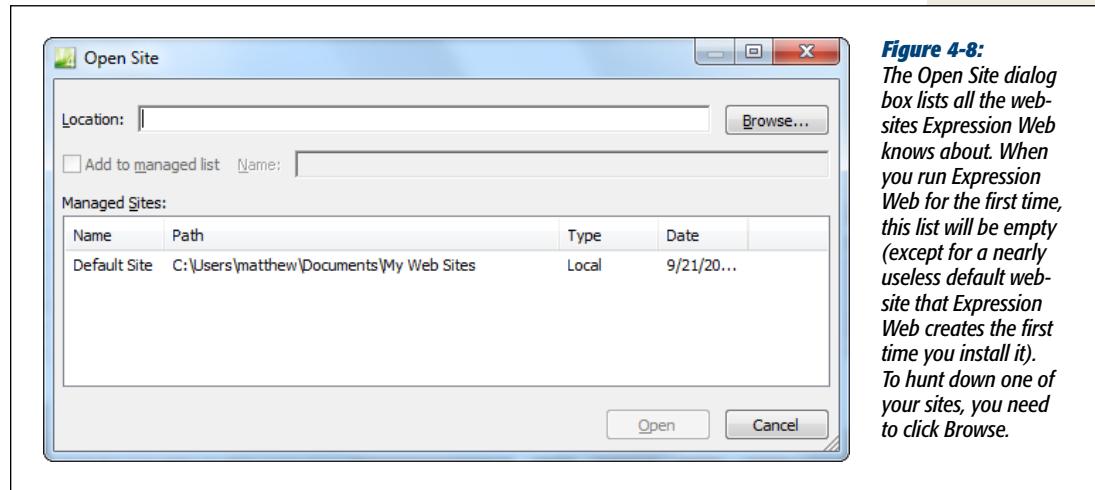
Note: A website is simply a collection of one or more web pages, along with any related files (like pictures). It's often useful to manage all these files together in a web page editor. In some programs, creating a website makes available advanced features, like templates, link checking, and website uploading.

Defining a Site in Expression Web

To create a website in Expression Web, follow these steps:

1. Select Site→Open Site.
The Open Site dialog box appears (Figure 4-8).
2. Click Browse to search for your website folder.
Opening a website is just like browsing for a file, except that you'll see only folders listed, not file names.
3. Browse to the folder you want to open, select it, and then click Open.
You can download the Chapter 2 folder from the Missing CD page (www.missingmanuals.com/cds/caw3) to get a ready-made folder and set of files (the résumé files you worked on earlier).

Clicking Open returns you to the Open Site dialog box. If you want to store the location of this website so you can open it more quickly next time, click “Add to managed list.” Once you do, Expression Web adds the folder to its Managed Sites list.

**Figure 4-8:**

The Open Site dialog box lists all the websites Expression Web knows about. When you run Expression Web for the first time, this list will be empty (except for a nearly useless default website that Expression Web creates the first time you install it). To hunt down one of your sites, you need to click Browse.

4. Click Open to load your website into Expression Web.

After you click Open, the editor displays a Site View tab listing all the files in your site (see Figure 4-9).

5. Add the Expression Web metadata folders. To do so, choose Site→Site Settings, choose “Maintain the website using hidden metadata files,” and then click OK.

Many of Expression Web’s site-management features require tracking information, which Expression Web stores in hidden subfolders. However, Expression Web doesn’t create these folders automatically; you need to opt in to get the program to produce them.

The word *metadata* means “data *about* data.” In other words, Expression Web’s metadata folders store data *about* the data in your website. If you’re curious, you can see these subfolders in Windows Explorer—they have names like *_private*, *_vti_cnf*, and *_vti_pvt*. (Web trivia: The VTI acronym stands for Vermeer Technologies Inc.—the company that originally created FrontPage and sold it to Microsoft.)

These folders have several purposes. First, they keep track of what files you uploaded to your web server. This tracking makes it incredibly easy for you to update a website, because Expression Web transfers only changed files to your server, not the entire site. The folders also track information about your site’s pages and resources, which Expression Web uses for handy features like link-checking (page 232) and templates (page 275).

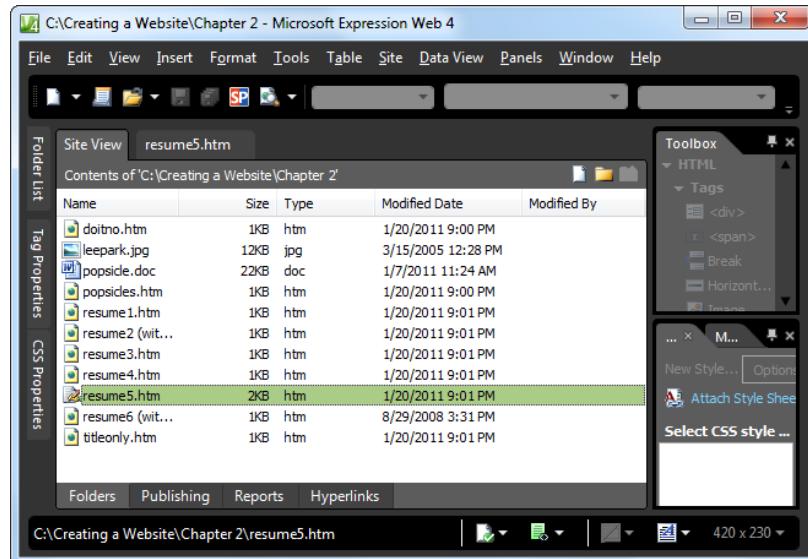


Figure 4-9: When you open a website folder (here, it's a folder named C:\Creating a Website\Chapter 2), Expression Web adds a tab that displays all the files in that folder. You can do basic file management here—for example, you can right-click a file to pop open a menu with options to rename or delete the file. You can also double-click a page to open it for editing. At the bottom of the Site View tab, you'll see buttons that let you publish your website, run a report, or check your links.

Tip: Treat the metadata folders as a bit of behind-the-scenes plumbing. You need to have them for certain features, but once you create them you don't need to think about them again.

Uploading a Site in Expression Web

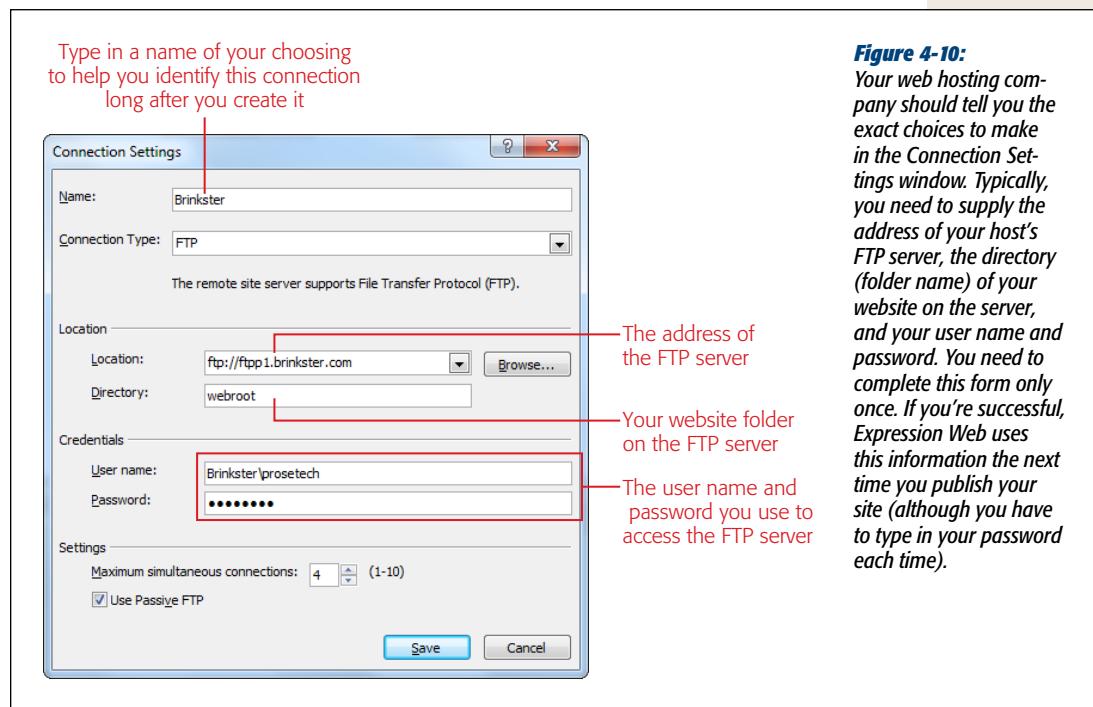
One of Expression Web's most popular features lets you update your website without the need for a separate FTP program to transfer site files to your server. Before you can take advantage of this, you need to follow the steps described above to tell Expression Web that your folder of web pages represents a complete website. Then, you can update your site with these steps:

1. Choose Site→Publishing (or click Publishing at the bottom of the Site View tab).

You can't publish your website until you fill in some basic information about your web hosting company,

2. Click the big “Add a publishing destination” link in the middle of the Site View tab.

The Connection Settings window appears (see Figure 4-10).



- Fill in the information that tells Expression Web how to connect to your server, and then click Add.

Expression Web saves the info and connects to your web host. The next time you use Expression Web's publishing feature, you'll see a "Connect to publishing destination" link instead of the "Add a publishing destination" link. Just click once to connect, with no extra work.

Once you're connected, Expression Web shows you a side-by-side file list that compares the contents of the website stored on your computer with that located on the server, so you can tell at a glance which files have changed (Figure 4-11).

- To bring your web server up to date, choose Site→Publish Changed Files.

This choice starts the publishing process (see Figure 4-12).

You can also transfer individual files using the arrows that appear between the two file lists. To transfer files from your computer to the server, select them on the left list and click the right-facing arrow. To download a file from your server, select it from the right list and click the left-facing arrow. The two-way arrow underneath ("Synchronize files") is like both operations rolled into one; it examines each file you select and makes sure it updates any old versions on either your computer or web server.

Figure 4-10:
Your web hosting company should tell you the exact choices to make in the Connection Settings window. Typically, you need to supply the address of your host's FTP server, the directory (folder name) of your website on the server, and your user name and password. You need to complete this form only once. If you're successful, Expression Web uses this information the next time you publish your site (although you have to type in your password each time).

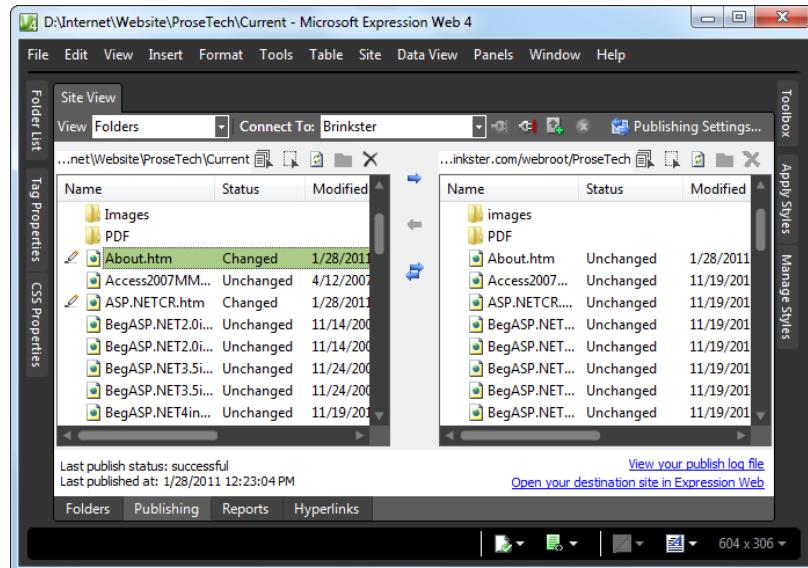


Figure 4-11:
Here, Expression Web highlights two files that you updated on your computer, but not on the web server: About.html and ASP.NETCR.htm.

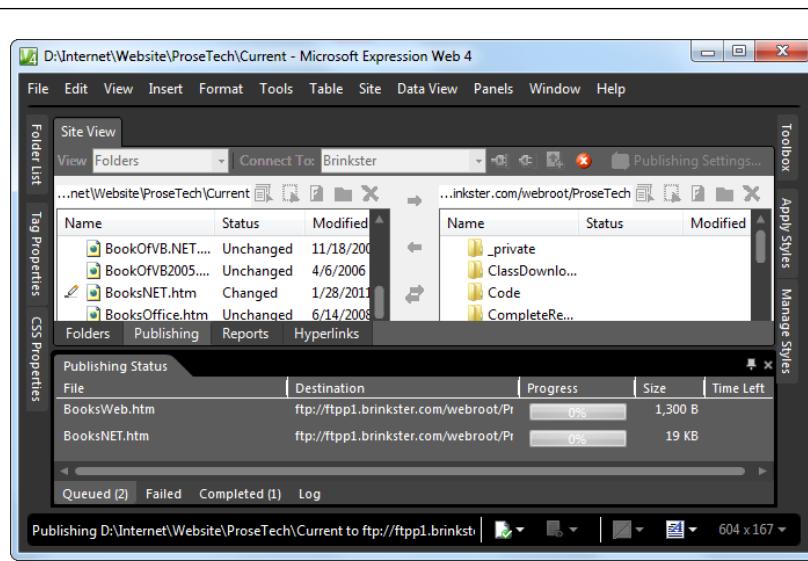


Figure 4-12:
When you publish a website, Expression Web scans your files and copies only the ones you added or changed since the last time you published the site. A progress indicator identifies the file being copied and estimates how long the operation will take.

Defining a Site in Dreamweaver

Dreamweaver gives you two ways to work with a website. The simplest is to use the Files panel to look at the files in any folder on your computer (see Figure 4-13).

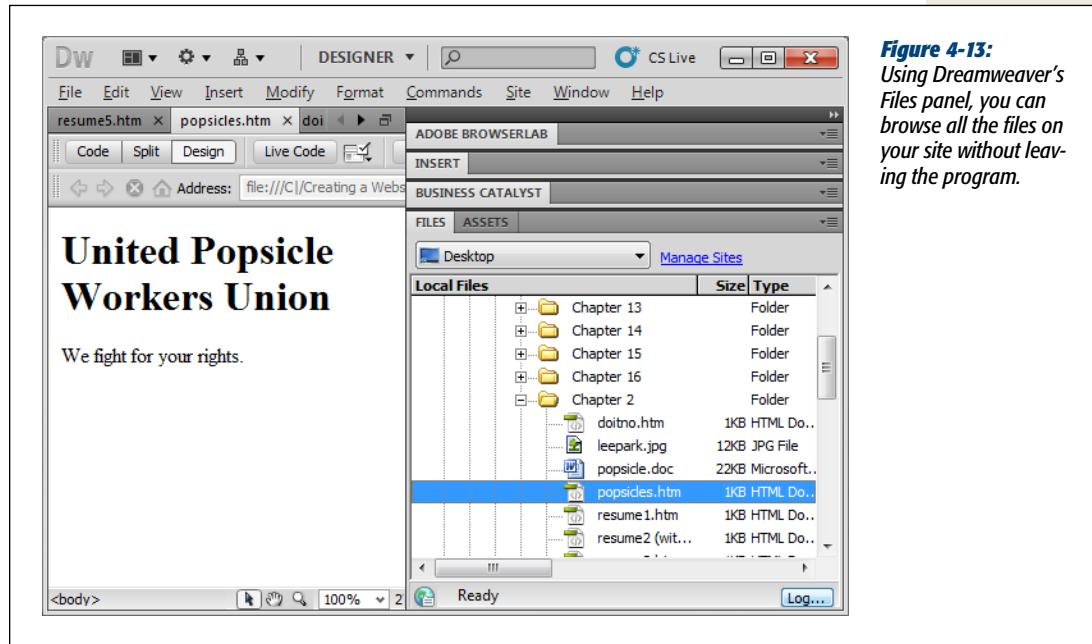


Figure 4-13:
Using Dreamweaver's Files panel, you can browse all the files on your site without leaving the program.

Although using the Files panel is convenient, it's also limiting. The problem is that Dreamweaver can't tell what folders and files make up your website. To support website uploading and a few other tools, you need to define a folder as a website. Here's how:

1. Click the Manage Sites link in the Files panel, or just select Site→Manage Sites.

Dreamweaver displays the Manage Sites dialog box, which lists all the websites you've configured so far. Initially, this list is empty.

2. To define a new site, click New.

Dreamweaver fires up a Site Setup window, where you fill in information about your new site.

3. Enter a descriptive name in the Site Name box.

The site name is just the one you use to keep track of your site, so use whatever name you want. This example uses LeeParkSite. The site name also appears in the Files panel.

4. In the Local Site Folder box, fill in the full file path for your website folder (usually something like C:\Creating a Website\Chapter 2).

If you aren't sure where your site folder is, you can click the folder icon next to the text box to browse for it. Or, you can enter a path to a folder that doesn't exist yet; Dreamweaver creates a new, empty folder so you can start building your new website.

5. In the list on the left, click Servers. Then, click the tiny plus icon in the bottom-right corner of the web server list (Figure 4-14).

A dialog box appears where you fill in your connection information (see Figure 4-15).

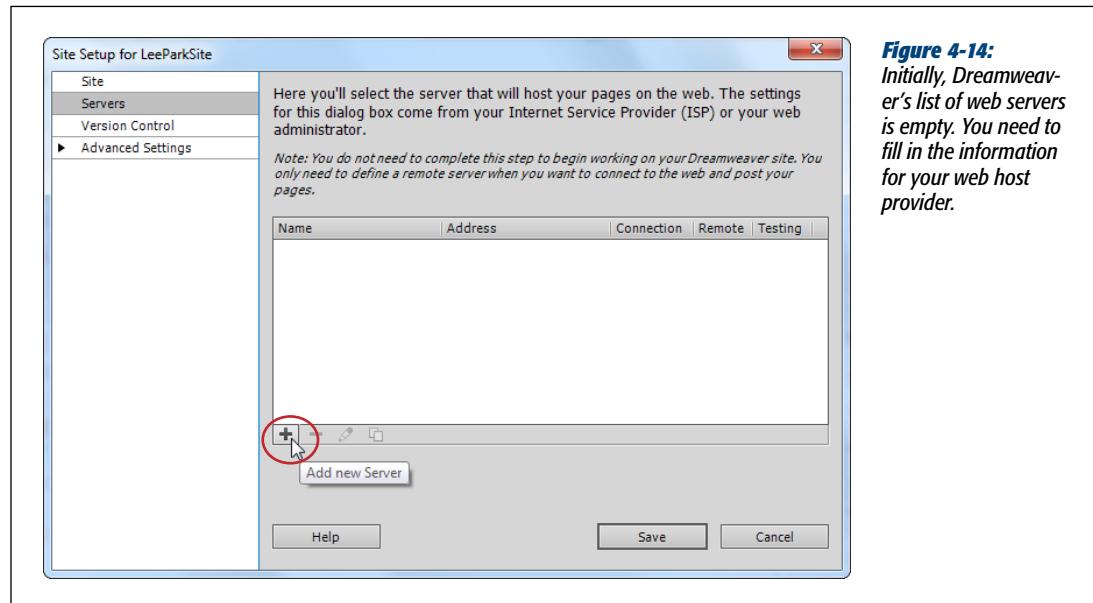


Figure 4-14:
Initially, Dreamweaver's list of web servers is empty. You need to fill in the information for your web host provider.

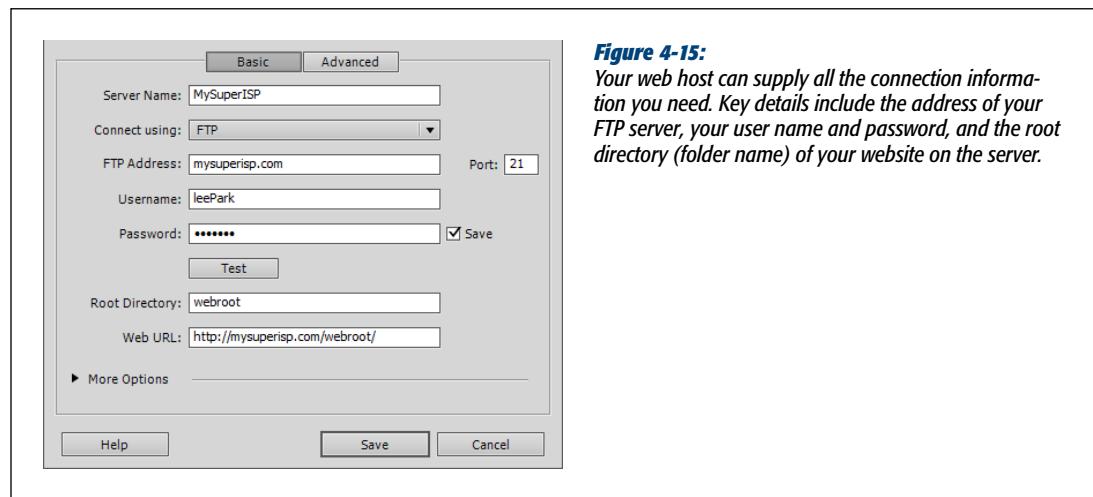


Figure 4-15:
Your web host can supply all the connection information you need. Key details include the address of your FTP server, your user name and password, and the root directory (folder name) of your website on the server.

6. When you finish entering your web server information, click Save.

You return to the Site Setup window. Dreamweaver stores your connection information so you won't need to enter it again.

7. Click Save again to close the Site Setup window.

You return to the Manage Sites dialog box.

8. Click Done.

You return to the Dreamweaver main window.

Uploading a Site in Dreamweaver

Once you define your website, you'll see it in the Files panel, and you can browse your remote web server for files, or transfer files back and forth from your computer to the server.

To transfer files from your local computer to your server, you use an operation that FTP jargon calls a *put*. It works like this:

1. In the Files panel, choose your website from the drop-down menu at the top left.
2. Choose “Local view” from the drop-down menu at the top right.

The Files panel lists the files on your computer (see Figure 4-16).

Tip: Initially, the file view is crammed into a small corner of the Dreamweaver window. If you want to expand it to fill the entire window, and show both the files on your local hard drive *and* those on the web server, click the expand button shown in Figure 4-16. This gives you a view that's similar to the Site View tab in Expression Web.

1. Select the files you want to transfer to the web server.

You can select multiple files by holding down Ctrl (⌘) while you click each file's icon.

2. Click the Put arrow (the blue arrow icon pointing up), or right-click the files, and then choose Put from the drop-down menu.

Dreamweaver asks if you want to copy dependent files.

3. Choose Yes if you want to copy all the files linked to your site pages. For example, if you upload a page that uses `` elements to display graphics, click Yes to make sure Dreamweaver uploads the graphics files as well as the pages themselves. If you don't have any dependent files, your choice has no effect.

Dreamweaver connects to your web server and transfers the files.

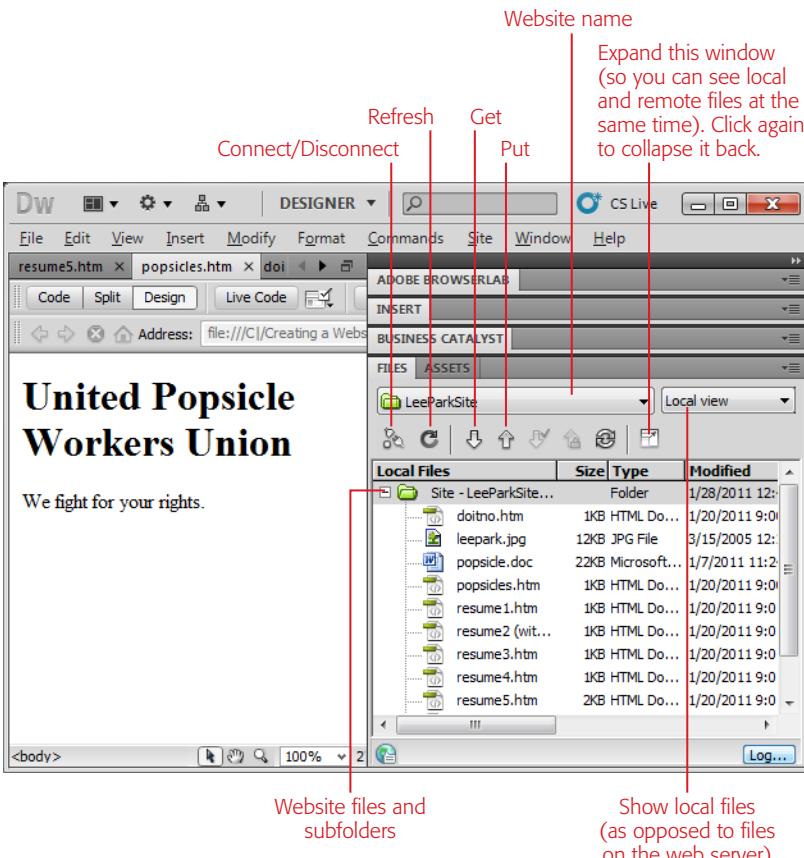


Figure 4-16:
This is the local view of a website named LeeParkSite. It lists all the files in the site folder on your computer. Using the icons in this window, you can quickly transfer files to and from your server.

To perform the reverse trick and transfer files from your server to your computer, you use a get operation. Follow these steps:

1. In the Files panel, choose your website from the drop-down menu at the top left.
2. Choose “Remote view” from the drop-down menu at the top right.

Dreamweaver doesn’t automatically display the list of files on your server, because getting that list could take a little time. So you need to specifically ask Dreamweaver for an updated view of the files on the server, which you’ll learn to do in the next step.

3. Click the Refresh button, which looks like a circular arrow icon.

Dreamweaver connects to the web server, retrieves the list of files on your site, and displays it.

4. Select the files you want to transfer to your computer.

You can select multiple files by holding down Ctrl (⌘) while you click each file's name.

5. Once you select the files, click the Get arrow (the green arrow icon pointing down), or right-click the files, and then choose Get from the drop-down menu.

Dreamweaver asks if you want to copy dependent files.

6. Choose Yes if you want to copy linked files. For example, if you download a page that uses `` elements to display graphics, click Yes to make sure Dreamweaver also downloads the graphics. If your page doesn't have any dependent files, your choice has no effect.

Dreamweaver connects to your web server and copies the files to the site folder on your computer.

Tip: Once you're comfortable transferring small batches of files, you can try out Dreamweaver's Synchronize button. It works like the website publishing feature in Expression Web. When you click it, Dreamweaver examines the web page files on your computer, determines which ones you updated, and transfers just those to your server.

Text Elements

Getting text into a web page is easy. You just open up an HTML file, drop in your content, and add the occasional HTML tag to format that content. But getting text to look *exactly* the way you want is a different story.

Before you can make your web pages look pretty, you need to put in a bit of organization work. In this chapter, you'll get the first tool you need: the HTML elements that let you break masses of text into neatly separated headings, paragraphs, lists, and more. You put several of these elements at work in the résumé examples in Chapter 2, but now you'll tour the HTML workshop to see everything it has to offer. Once you finish this step, you'll be ready for the next chapter, where you learn to format your carefully structured content with *style sheets*—powerful page-formatting instructions that let you change the appearance of individual elements, entire web pages, and even your complete website.

Understanding Text and the Web

Sooner or later, every website creator discovers that designing for the Web is very different from designing something that's going to be printed. Before you can unleash your inner web-page graphic designer, you need to clear a few conceptual hurdles.

Consider the difference between an HTML page and a page created in a word processor. Word processing programs show you exactly how a document will look before you print it: You know how large your headlines will be, what font they're in, where your text wraps from one line to the next, and so on. If you see something you don't like, you change it using menus and formatting commands. Your word processor, in other words, gives you absolute control over every detail of your page.

The Web is a more freewheeling place. When you create an HTML document, you know some—but not all—of the details about how that web page will appear on someone else’s computer. While one person’s browser may display large type instead of standard-size characters and stretch it to fill the full expanse of a 28-inch widescreen monitor, another’s may be shrunk and tucked it away in a corner of the desktop. And visitors who access your site with a smartphone, tablet computer, or web-enabled toaster will get yet another—and completely different—view. In short, you can’t micromanage display details on the Web the way you can in print. But you *can* supply all the information a web browser needs to present your pages properly. You do this by structuring your pages so that browsers treat your page elements consistently, regardless of your visitors’ browser settings.

Logical Structure vs. Physical Formatting

Before you start creating web pages with HTML markup, you should understand one other concept—the difference between *structuring* a document (dividing its content into discrete components like headings, paragraphs, lists, and so on) and *formatting* a document (making those components look pretty by applying italics, changing the text size, adding color, and so on). Novice web masters who don’t understand this difference often end up formatting when they should be structuring, which leads to messy and difficult-to-maintain HTML pages.

In the early days of the Web, HTML used two different types of elements to emphasize the distinction between document structure and document formatting:

- **Logical elements** (also called *semantic elements*) define the individual components that make up your web page. They identify what in a page is a heading, a paragraph, a list, and so on. In other words, they tell you about the *structure* of your page. Logical elements don’t, however, define the *format* of those components —what the components will look like once they’re onscreen.
- **Presentational elements** (also called *typographic elements*) are all about formatting. Examples include elements that apply italics, boldface, underlining, and font settings to text. Presentational elements don’t tell you anything about the way your page is structured; instead, they tell browsers how to *format* the content in those pages.

Note: You got a taste of both of these element types in the résumé example in Chapter 2. Logical elements created the heading, paragraphs, and bulleted list that structured the résumé. The presentational elements added bold and italicized effects to small portions of the text.

Logical elements have ruled the markup roost since HTML was invented. XHTML 1.0 furthered the dominance of logical elements by removing many of its presentational elements (like ``, for changing the typeface, and `<u>`, for underlining text). HTML5 goes even further—it changes the definition of two presentational elements that stubborn web developers refused to give up: `` and `<i>`. Now, `` no

longer means “bold text” but “stylistically offset text” and `<i>` no longer means “italic text” but “text in an alternate voice, such as foreign words and technical terms.” Of course, web browsers aren’t affected. They always render `` text in bold and `<i>` text in italics, unless you use a style sheet that tells them not to. But the change in wording emphasizes a fundamental shift in philosophy—namely, that HTML elements should describe the logical function of portions of text, not the typographic presentation of it.

At this point, you’re probably wondering why HTML gurus were so keen to strip the formatting out of HTML documents. There are a few good reasons:

- **Embedded formatting is messy.** After you see the markup gunk in an old HTML page, you never want to return. To make even simple edits, you need to wade through the swamp of formatting tags that controls margins, colors, font, and alignment.
- **Logical documents are adaptable.** By taking the formatting out of your page, your page becomes much more flexible. You can use style sheets (see the following section) to change the way your web page looks, and you change them whenever you want. And if you need to display the web page in a different way on a different sort of web-enabled device, you (or a browser) can do that too.
- **Logical documents carry meaning.** Logical elements let external programs analyze a page’s HTML. For example, someone could create an automated search program that scanned web pages and extracted just the top-level headings to produce a barebones outline of the page. Or a program could browse Amazon .com to find only book reviews. Or one could create a junk-mail list by reading `<address>` elements. A comparable program that came across a web page filled with nothing but presentational elements wouldn’t produce results nearly as interesting. After all, who cares how much of eBay’s text is in boldface?
- **Logical documents are accessible.** Screen-reading software and other automated tools work far better with logically structured pages. For example, a screen-reading program can guide a visually impaired person around your page by announcing headings or links, rather than reading everything out loud.

Note: The vision of a Web where elements indicate *what* a page contains (prices, size information, email addresses, and so on) rather than *how* it looks is called the *Semantic Web*. According to the visionaries who first built the Internet, the Semantic Web could usher in a golden age of information access and super-smart searching.

CSS (Cascading Style Sheets)

All of this discussion about page structure raises a good question: If, in developing the “cleanest” pages possible, you’re supposed to use elements just to set out the structure of your pages, what’s left to make those pages look good?

For example, say you want to give all your headings hot pink lettering. If HTML had an element to specify text color, you'd need to include that element every time you placed a heading on a page. Now imagine that you later decide to go with a more refined dark purple heading. To change your website, you'd have to open and edit each and every HTML page that had a heading. Consider how much easier life would be if you could link your headings to a formatting instruction in a central reference page. With this kind of setup, you could change the color of the headings across your entire site by simply changing the formatting instruction for headings in that one central reference page.

That's the solution that HTML experts finally hit on. They separated a document's structure—the elements that create headings, paragraphs, lists, and so on—from the formatting instructions, and placed each part in a separate file. Here's how it works. First, you create standard HTML documents just like the ones you learned to create in Chapter 2. These documents include the same elements for headings, paragraphs, and lists that you learned about earlier. That's good news—it means you don't need to change your approach or throw out the basic elements you've already mastered.

Next, you create a separate document using a standard called CSS (Cascading Style Sheets). This document is called a *style sheet*, and it defines how browsers format the different elements in your HTML document. For example, a style sheet might contain instructions like “make every heading bright red” or “give all paragraphs a 15-pixel left margin.”

The style sheet system offers many benefits. First of all, you can reuse the same style sheet for all your web pages. Because getting your formatting right can be a long and tedious chore, this is a major timesaver. Once you perfect your website's look and feel, you link your pages to this style sheet and all your pages take on that design (see Figure 5-1). Even better, when you're ready for a new look, you don't need to mess with your HTML documents—just tweak your style sheet and every linked page gets an instant facelift.

Note: HTML still has a few formatting features that date back to the bad old days of the Web. They aren't as powerful as style sheets, and they're a lot messier. Now that you're thinking with style sheets in mind, you're ready to steer clear of those headaches and concentrate on becoming comfortable with the staples of the HTML diet—the elements for structuring text.

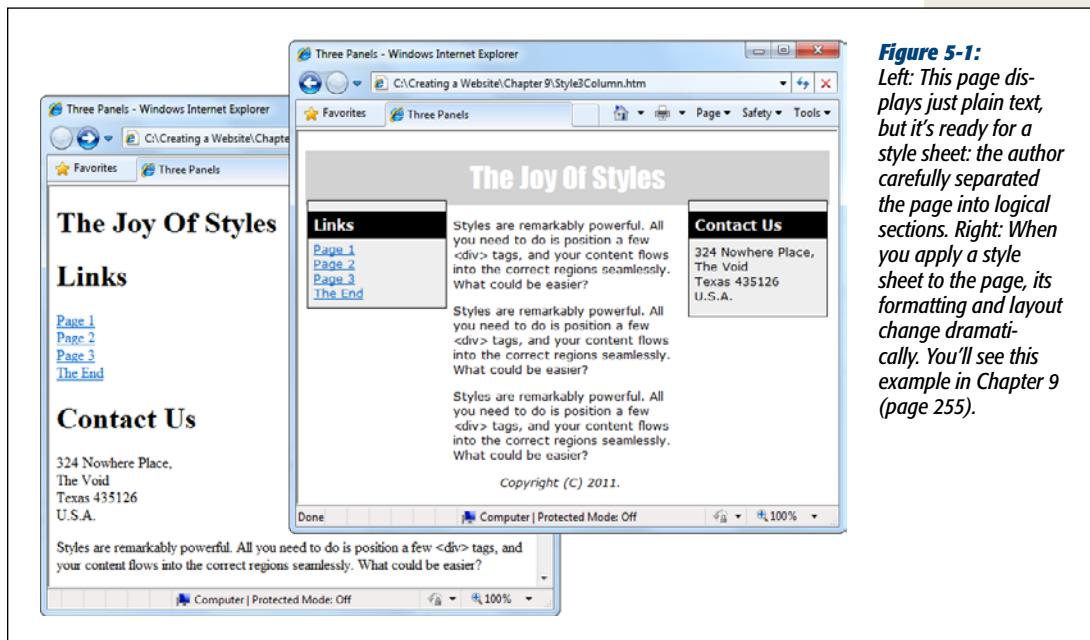


Figure 5-1:
Left: This page displays just plain text, but it's ready for a style sheet: the author carefully separated the page into logical sections. Right: When you apply a style sheet to the page, its formatting and layout change dramatically. You'll see this example in Chapter 9 (page 255).

HTML Elements for Basic Text

As you learned in Chapter 2, you need to know two things about every new element you meet. To use an element correctly, you need to answer these two questions:

- Is it a container element or a standalone element?
- Is it a block element or an inline element?

The first question tells you something about the syntax you use when you add an element to a document. Container elements (like the `` element that boldfaces text) require a start and end tag, with the content sandwiched in between. Standalone elements (like the `` element that inserts an image into a page) use a single, all-in-one tag. If standalone elements need additional information, like the location of an image file, you supply it using attributes.

The second question tells you something about *where* you can place an element. Block elements (like the `<p>` element) go inside the main `<body>` element or within other block elements of a page. When you start building the overall structure of your page, you always begin with block elements. Inline elements (like the `` element) have to go *inside* block elements. Inline elements don't make sense when they're on their own, floating free of any container.

Tip: To quickly check whether an element is a container or standalone element and to see if it's a block or inline element, check the HTML reference in Appendix A.

Block elements also have an effect on the spacing of your content. Essentially, each block element defines a chunk of content. When you end a block element, your browser automatically adds a line break and a little extra space before the next bit of content.

For example, consider this fragment of HTML:

```
<h1>Bread and Water</h1><p>This economical snack is really  
all you need to sustain life.</p>
```

This snippet has a title in large, bold letters followed immediately by a paragraph of ordinary text. You might expect to see both parts (the heading and the ordinary text) on the same line. However, the `<h1>` element is a block element. When you close it with the `</h1>` tag, the browser does a little housecleaning, and adds a line break and some extra space before the next element. The paragraph text starts on a new line, as you can see in Figure 5-2.

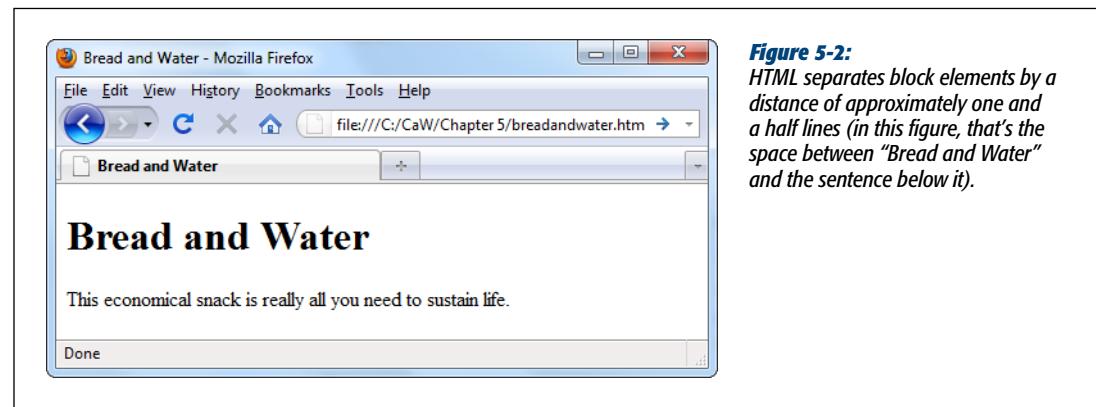


Figure 5-2:
HTML separates block elements by a distance of approximately one and a half lines (in this figure, that's the space between "Bread and Water" and the sentence below it).

Browsers space out block elements to help ensure that one section of text doesn't run into another. However, in many cases, you won't be happy with this automatic spacing. For example, for dense, information-laden pages, the standard spacing looks far too generous. To tighten up your text and shrink the spaces in between block elements, you can use style sheets to change the elements' margin settings (page 153).

In an attempt to talk less about formatting and more about structure, HTML5 proposes some new terminology. It suggests that inline elements be called *phrasing elements* and block elements be called *flow elements*. The goal is to emphasize the difference in the way you can use and place these types of elements, while downplaying the way they affect formatting. (After all, the formatting details can be changed

with style sheets.) However, the terms “block element” and “inline element” are so common that it will be a while before new lingo replaces them—if the HTML5 jargon catches on at all.

Now that you’ve learned about the basic types of elements, it’s time to take a look through your element toolkit.

Paragraphs

You’ve already seen the basic paragraph element, `<p>`. It’s a block element that defines a paragraph of text.

```
<p>It was the best of times, it was the worst of times...</p>
```

As you’ve no doubt noticed in your travels across the Internet, HTML paragraphs aren’t indented as they are in print media. That’s just the way of the Web, although you can change this with style sheets (page 153). Figure 5-3 shows an example of paragraph elements in action.

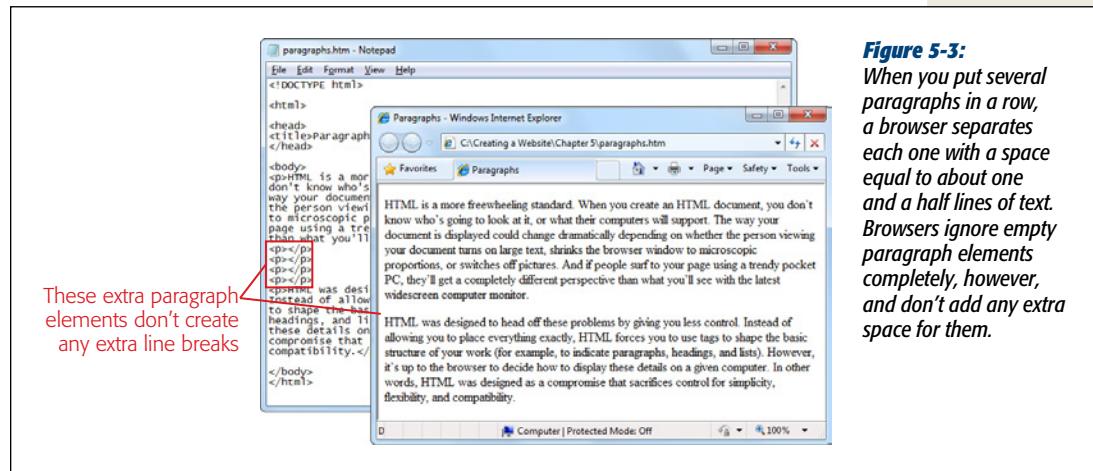


Figure 5-3:
When you put several paragraphs in a row, a browser separates each one with a space equal to about one and a half lines of text. Browsers ignore empty paragraph elements completely, however, and don't add any extra space for them.

You should get into the habit of thinking of the text in your web pages as a series of paragraphs. In other words, before you type in any text, add the `<p>` and `</p>` tags to serve as a container. Most of the time, paragraphs are the first level of structure you add to a page.

Web browsers don’t pay attention to *hard returns* (the line breaks you create when you hit the Enter key). That means you can space your text out over several lines as you type it in, and the browser still wraps it to fit the window. Technically, browsers treat line breaks (like the one you see at the end of this line) as a single space. If a browser finds more than one space in a row, it ignores the extra ones. If you want to insert a *real* break between your lines, check out the next section.

UP TO SPEED

Getting More Space

The way that browsers ignore spaces can be exasperating. What if you really *do* want several spaces in a row? The trick is the nonbreaking space— —which is a special HTML character entity (see page 130) that forces browsers to insert a space.

When a browser sees this entity, it interprets it as a space that it can't ignore. So if you create a paragraph like this:

```
<p>Hello&nbsp;&nbsp;&nbsp;Bye</p>
```

You end up with this text:

Hello Bye

Many web editors automatically add nonbreaking spaces when you press the space key in Design view, which is why those spaces don't disappear. Try not to use nonbreaking spaces more than necessary. (If you really want indented paragraphs, you'll get a better solution with style sheets, which you'll learn about in Chapter 6.) And never, ever use spaces to align columns of text—that always ends badly, with the browser scrambling your attempts. Instead, use the layout features described in Chapter 9.

Line Breaks

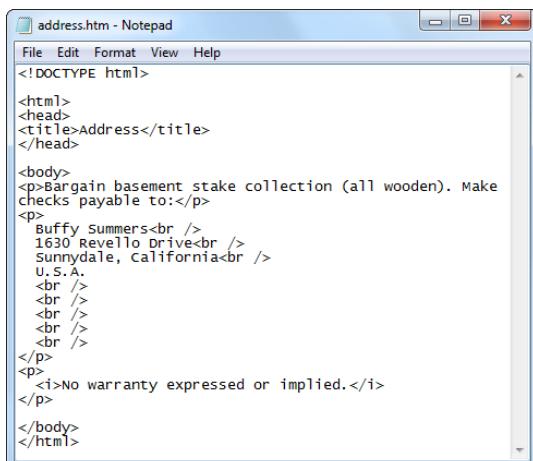
Sometimes you want to start a new line of text, but you don't want to use a paragraph element because browsers add extra space between paragraphs. This is the case, for example, when you want to include a business address on your site and you want it to appear in the standard single-spaced three-line format. In situations like this, the standalone line break element `
` comes in handy.

Line breaks are exceedingly simple: They tell a browser to move to the start of the following line (see Figure 5-4). They're inline elements, so you need to use them inside a block element, like a paragraph:

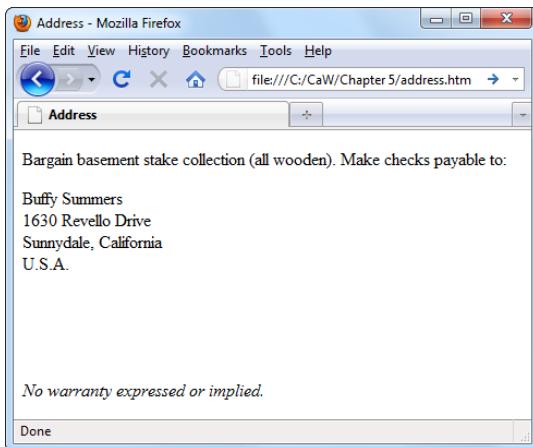
```
<p>This paragraph appears<br />  
on two lines</p>
```

Don't overuse line breaks. Remember, when you resize a browser window, the browser reformats your text to fit the available space. If you try to perfect your paragraphs with line breaks, you'll end up with pages that look bizarre at different browser window sizes. A good rule of thumb is to avoid line breaks in ordinary paragraphs. Instead, use them to force breaks in addresses, outlines, poems, and other types of text whose spacing you want to tightly control. Don't use them for bulleted or numbered lists, either. You'll learn about elements designed just for these on page 119.

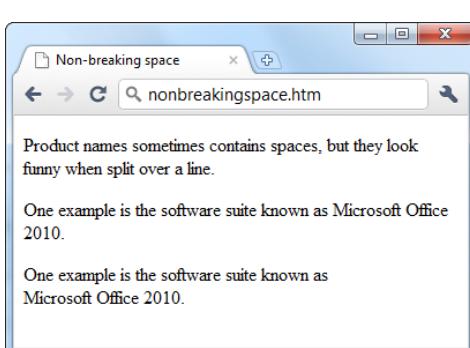
In some cases, you want to *prevent* a line break, like when you want to keep the longish name of a company or a product on a single line. The solution is to use the nonbreaking space code (which looks like) instead of just hitting the space bar. The browser still displays a space when it gets to the code, but it won't wrap the words on either side of it (see Figure 5-5).



```
<!DOCTYPE html>
<html>
<head>
<title>Address</title>
</head>
<body>
<p>Bargain basement stake collection (all wooden). Make checks payable to:</p>
<p>Buffy Summers<br />
1630 Revello Drive<br />
Sunnydale, California<br />
U.S.A.<br />
<br />
<br />
<br />
<br /><br />
</p>
<p><i>No warranty expressed or implied.</i></p>
</body>
</html>
```

**Figure 5-4:**

The line break element `
` is great for separating lines in an address. If you want to skip down several lines, you can use a series of `
` elements in a row (but it's a better idea to use empty paragraphs, as described in the box on page 112).

**Figure 5-5:**

Paragraphs 2 and 3 in this figure show how the ` ` code affects line breaks. Paragraph 3 is actually coded as `Microsoft Office 2010`. As a result, the browser won't split this term.

HOW'D THEY DO THAT?

The Mystery of Empty Paragraphs

In web authoring tools like Dreamweaver and Expression Web, if you're in Design view and you press Enter, the program creates a new paragraph. This seems a little counter-intuitive, as you've seen that browsers normally ignore line breaks (see Figure 5-5).

The trick is that when you hit the Enter key, both programs insert a paragraph that contains a nonbreaking space. Here's what that creation looks like:

```
<p>&nbsp;</p>
```

This paragraph is still empty, but the browser won't ignore it because it includes the code. Therefore, the browser gives it the same space as a single-line paragraph and bumps down the content underneath.

Incidentally, Dreamweaver and Expression Web do let you use more ordinary
 line break elements instead of empty paragraphs, even in Design view. To do this, press Shift+Enter instead of Enter.

Headings

Headings are section titles—for example, the word “Headings” just above this paragraph. Browsers display them in boldface at various sizes, depending on the *heading level*. HTML supports six levels, starting at <h1> (the biggest) and dwindling down to <h6> (the smallest). Both <h5> and <h6> are smaller than regularly sized text. Figure 5-6 shows all the heading levels you can use.

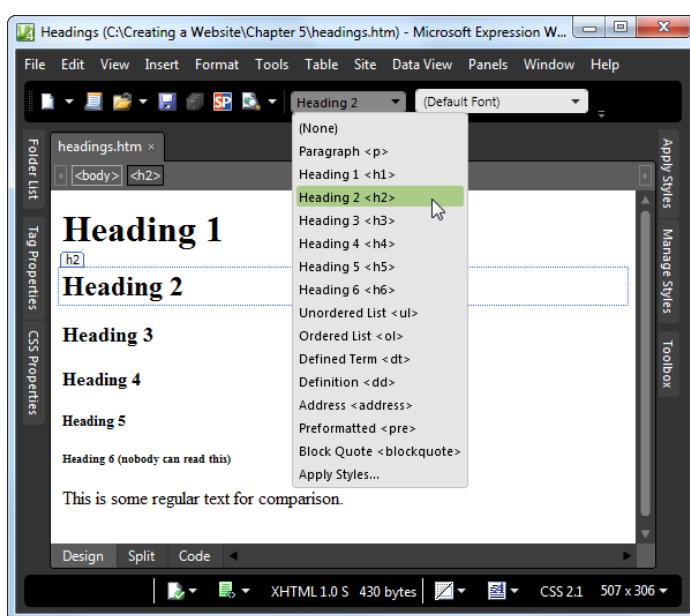


Figure 5-6:

Many web page editors let you apply headings with a single click. In Dreamweaver, you can use the handy buttons in the Text tab of the Insert toolbar. In Expression Web, you can find a drop-down list that lets you choose whether to make the currently selected text a paragraph or one of the various headings, as shown here.

Headings aren't just useful for formatting—they also help define the hierarchy of your document. Big headings identify important topics, while smaller ones denote lesser issues related to that larger topic. To make sure your document makes sense, start with the largest headings (level 1) and work your way down. For instance, don't jump straight to a level-3 heading just because you like the way it looks.

Note: It's probably occurred to you that if everyone uses the same heading levels in the same order, the Web will become as bland as a bagel in a chain supermarket. Don't panic—it's not as bad as it seems. When you add style sheets into the mix, you'll see that you can completely change the look of any and every heading you use. So for now, stick to using the right heading levels in the correct order.

Horizontal Lines

Paragraphs and line breaks aren't the only way to separate sections of text. Another neat trick is the standalone `<hr>` element, which translates to “horizontal rule.” A horizontal rule element adds a line that stretches from one side of its container to the other, separating everything above and below it.

Note: Usually, you position a horizontal break between paragraphs, which means it will stretch from one side of a page to the other. However, you can also put a horizontal rule in a smaller container, like a single cell in a table, in which case it won't turn out nearly as big.

Horizontal rules are block elements, so you can stick them in between paragraphs (see Figure 5-7).

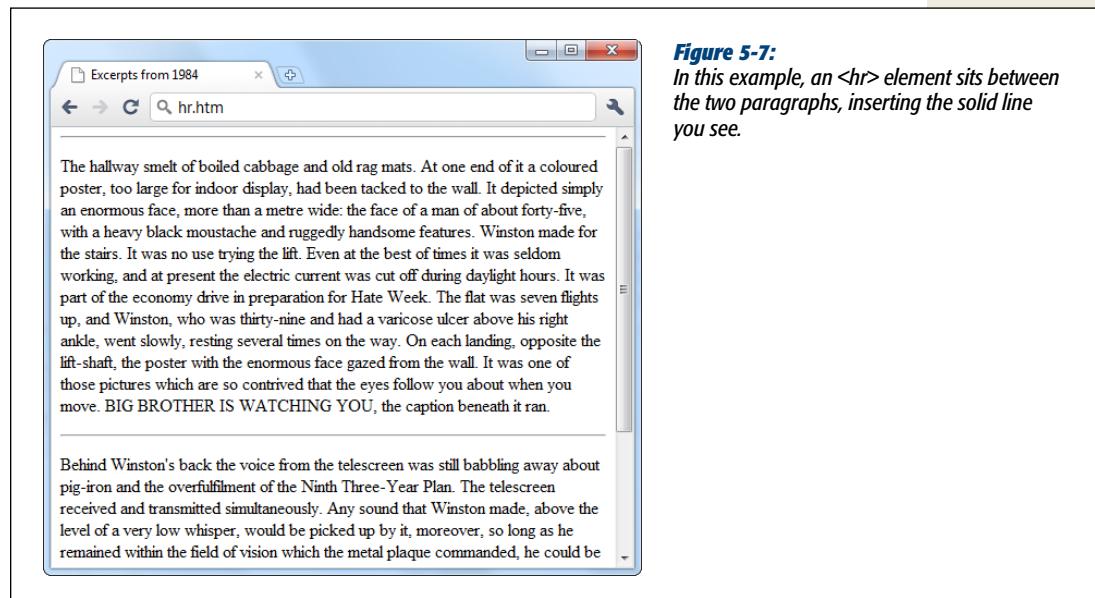


Figure 5-7:

In this example, an `<hr>` element sits between the two paragraphs, inserting the solid line you see.

Obviously, the `<hr>` element is a presentational element (page 104), because it's all about drawing a line in your page. The authors of the HTML5 specification try to make it more about logical structure by describing it not as a "horizontal rule" but as a "thematic break"—in other words, a basic way to separate different blocks of content, like scenes in a novel or topics in an academic paper. But it still looks like a line.

Preformatted Text

Preformatted text is a unique concept in HTML that breaks the rules you've read about so far. As you've seen, web browsers ignore multiple spaces and flow your text to fit the width of a browser window. Although you can change this to a certain extent using line breaks and nonbreaking spaces, some types of content are still hard to deal with.

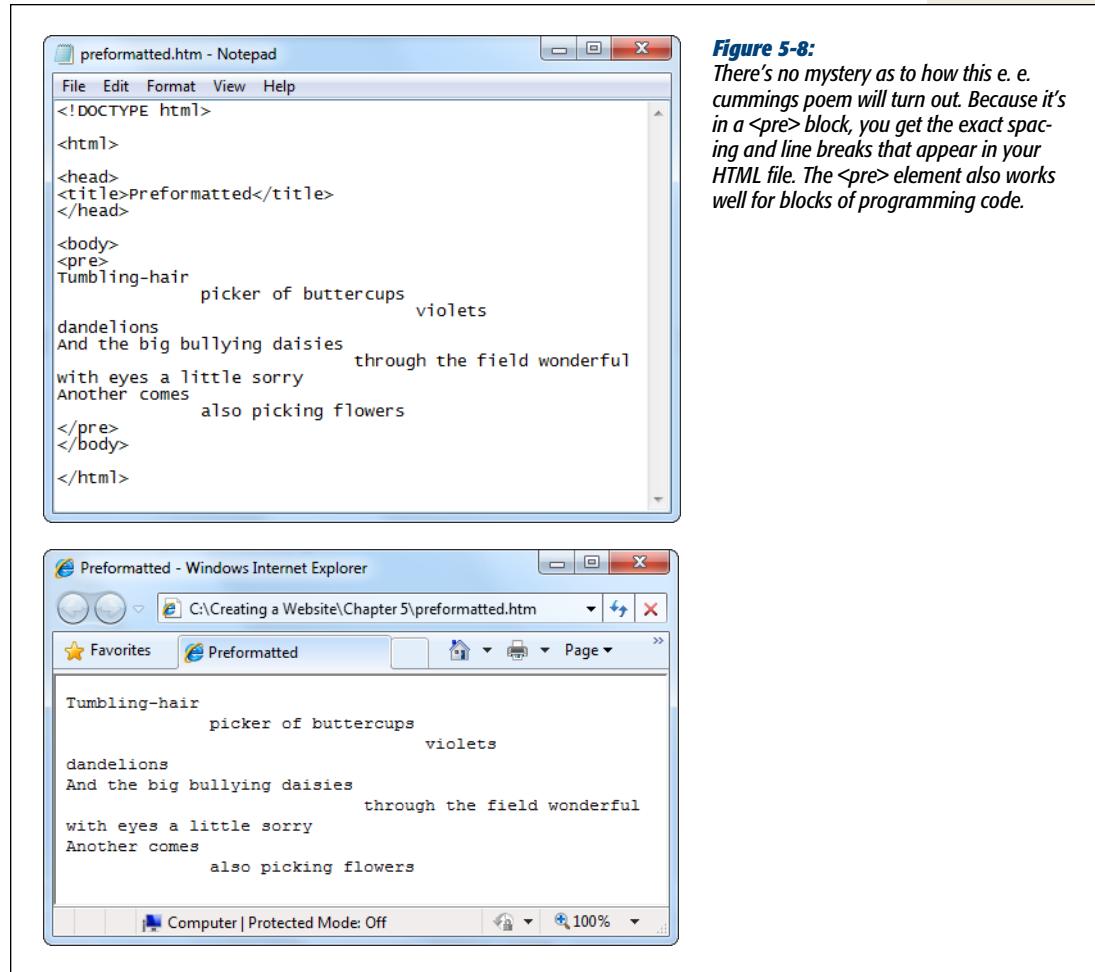
For example, imagine you want to display a bit of poetry. Using nonbreaking spaces to align the text is time-consuming and makes your HTML markup difficult to read. The `<pre>` element gives you a better option. It tells your browser to re-create the text just as you entered it, including every space and line break, and it displays these details onscreen. Additionally, the browser puts all that text into a monospaced font (typically Courier), further setting this content off from the rest of the page. Figure 5-8 shows an example.

Note: In a *monospaced* font, every letter occupies the same amount of space. HTML documents and books like this one use proportional fonts, where letters like W and M are much wider than l and i. Monospaced fonts are useful in preformatted text, because they let you line up rows of text exactly. The results, however, don't look as polished.

Sometimes web weavers use preformatted text when quoting computer code or the output that a computer program displays. In these cases, you can combine the `<pre>` element with the `<code>` element (to represent a snippet of code) or the confusingly named `<samp>` element (to represent "sample" computer output).

```
<p>This is an infinite loop in Visual Basic:  
<pre><code>  
Do  
    X = X + 1  
    Loop Until X < 0  
</code></pre>  
</p>
```

The `<code>` and the `<samp>` element apply a monospace font, just like the `<pre>` element. However, the `<pre>` element is the only one that tells the browser to include all the whitespace. That's why you need to use `<pre>` and `<code>` in combination.

**Figure 5-8:**

There's no mystery as to how this e.e. cummings poem will turn out. Because it's in a `<pre>` block, you get the exact spacing and line breaks that appear in your HTML file. The `<pre>` element also works well for blocks of programming code.

Quotes

It may be a rare web page that spouts literary quotes, but the architects of HTML created a block element named `<blockquote>` especially for long quotations. When you use this element, your browser indents text on the left and right edges.

Here's an example:

```
<p>Some words of wisdom from "A Tale of Two Cities":</p>
```

```
<blockquote>
```

```
<p>It was the best of times, it was the worst of times, it was the age of  
wisdom, it was the age of foolishness, it was the epoch of belief, it was  
the epoch of incredulity, it was the season of Light, it was the season of
```

```
Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way--in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.</p>
</blockquote>
```

```
<p>It's amazing what you can fit into one sentence...</p>
```

Figure 5-9 shows how this appears in a browser.

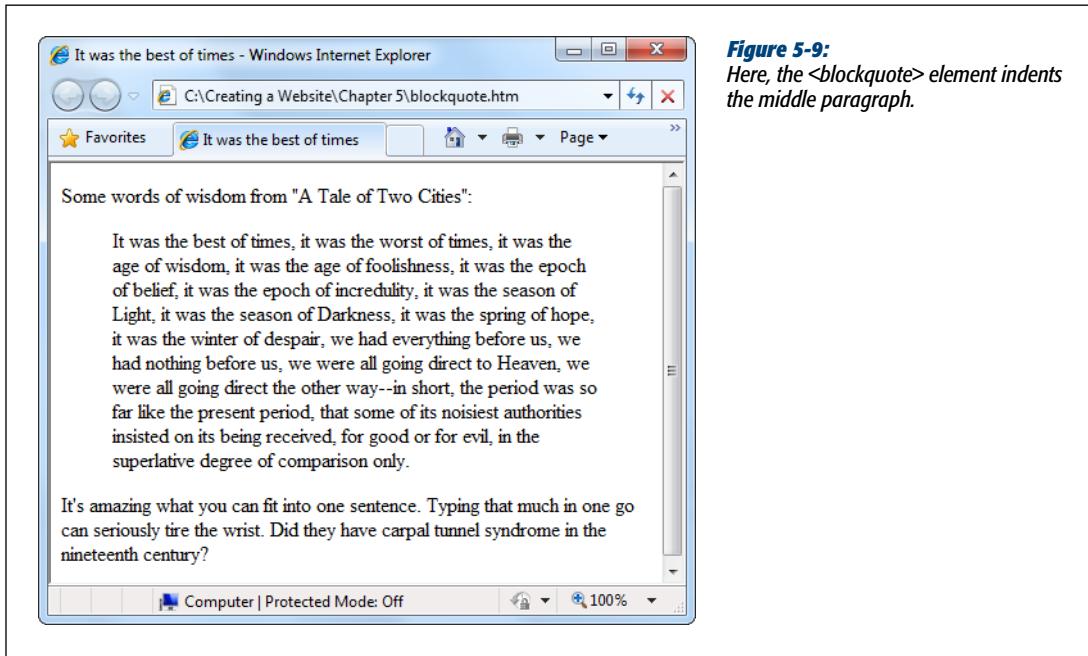


Figure 5-9:
Here, the `<blockquote>` element indents the middle paragraph.

Occasionally, people use the `<blockquote>` element purely for its formatting capability—they like the way it sets off text. Of course, this compromises the spirit of the element, and you'd be better off to use style sheets to achieve a similar effect.

The `<blockquote>` element is a block element, which means it always appears separately from other block elements, like paragraphs and headings. The `<blockquote>` element has one further restriction: It can hold only other block elements, which means you need to put your content into paragraphs rather than simply type it in between the blockquote start and end tags.

If, instead of using a quote that runs a paragraph or longer, you want to include a simple one-line quote, HTML's got you covered. It defines an inline element for short quotes that you can nest inside a block element. It's the `<q>` element, which stands for “quotation”:

```
<p>As Charles Dickens once wrote, <q>It was the best of times, it was the  
worst of times</q>.</p>
```

Some browsers, like Firefox, add quotation marks around the text in a `<q>` element. Other browsers, like Internet Explorer, do nothing. If you want your quotation to stand out from the text around it in every browser, you might want to add some different formatting, like italics. You can do this by applying a style sheet rule (see Chapter 6).

And if you want to pack more information into your quotes, you can add a URL that points to the source of the quote (assuming it's on the Web) using the `cite` attribute:

```
<p>As Charles Dickens once wrote,  
<q cite="http://www.literature.org/authors/dickens-charles/two-cities">It  
was the best of times, it was the worst of times</q>.</p>
```

Looking at this example, you might expect your browser to provide some sort of feature that takes you to the referenced website (for example, when you click the paragraph). But it doesn't. If you want your text to link to a reference, you need to investigate the anchor element in Chapter 8.

In fact, the information in the `cite` attribute won't appear on your page at all. It *is* available to programs that analyze your web page—for example, automated programs that scan pages and compile a list of references or a search engine that uses this information to provide better search results. Most of the time, however, the reference has little benefit, except that it stores an extra piece of information that you, the website creator, might need later to double-check your sources.

If you do reference a source in your text, you might want to use the `<cite>` element. It identifies the title of a work, like this:

```
<p>The quote <q>It was the best of times, it was the worst of times</q> is  
from <cite>A Tale of Two Cities</cite>.</p>
```

In this case, a browser displays the text in the `<cite>` element in italic, just as if you used the `<i>` element.

Divisions and Spans

The last block element you'll learn about—`<div>`—is one of the least interesting, at least at first glance. On its own, it doesn't actually *do* anything.

You use `<div>` to group together one or more block elements. That means you could group together several paragraphs, or a paragraph and a heading, and so on. Here's an example:

```
<div>  
  <h1>...</h1>  
  <p>...</p>  
  <p>...</p>  
</div>  
<p>...</p>
```

Given the fact that `<div>` doesn't do anything, you're probably wondering why it exists. The lowly `<div>` tag becomes a lot more interesting when you combine it with style sheets, because you can apply formatting commands directly to a `<div>` element. For example, if a `<div>` element contains three paragraphs, you can format all three paragraphs at once simply by formatting the `<div>` element.

Tip: As you'll see throughout this book, the `<div>` element is an indispensable, all-purpose container. You use it to shape the layout of your page, creating columns of text, navigation bars, floating figures, and more.

The `<div>` element has an important relative: the `` element. Like its cousin, the `` element doesn't do anything on its own, but when you place it *inside* a block element and define its attributes in a style sheet, you can use it to format just a portion of a paragraph, which is very handy. Here's an example:

```
<p>In this paragraph, some of the text is wrapped in a span element. That<br><span>gives you the ability</span> to format it in some fancy way later on.</p>
```

You'll put the `<div>` and `` elements to good use in later chapters.

HTML5 PREVIEW

Supercharged Structure

The `<div>` element is supremely flexible, but it doesn't say much about the real structure of your page. For example, one `<div>` element might represent a header that sits at the top of your page. Another might wrap a set of navigation links. For this reason, the `<div>` isn't much help to anyone (or any program) trying to scan your markup and figure out what it all means. While this isn't a fatal flaw in the `<div>` element, it is a nagging shortcoming.

As you learned on page 105, part of the dream for the future of the Internet is to embed structural details into your pages, so that browsers, search engines, and automated tools can understand their structure and find important bits of information. To make the task easier, HTML5 offers several new elements that you can substitute for a plain vanilla `<div>`, including `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>`, and `<nav>`. Like the `<div>` element, these elements have no built-in formatting. They're simply containers where you can place content (and then position

and format it with style sheets, as you'll learn in Chapter 6). But unlike the `<div>` element, these elements carry a bit more semantic meaning. For that reason, HTML5 purists prefer to put a group of navigation links in a `<nav>` element rather than a generic `<div>` element, even though the final result in the browser looks the same.

Of course, because the HTML5 elements are new, not all browsers can interpret them. This isn't a huge problem—after all, the `<div>`-replacement elements don't apply any formatting anyway, so it's perfectly fine if a browser chooses to ignore them. But there's a wrinkle with Internet Explorer, which doesn't let you use styles with unrecognized elements. To fix this problem, you need to use a JavaScript workaround. Appendix A describes this workaround (page 535) and details each of the HTML5 semantic elements. But people who don't need to be on the bleeding edge may prefer to wait for a while and stick to the tried-and-true `<div>`.

HTML Elements for Lists

Once you master HTML's basic text elements, you can tackle HTML's list elements. HTML lets you create three types of list:

- **Ordered lists** give each item in a list a sequential number (as in 1, 2, 3). They're handy when sequence is important, like when you list a series of steps that tell your relatives how to drive to your house.
- **Unordered lists** are also known as bulleted lists, because a bullet appears before each item. To some degree, you can control what the bullet looks like. You're reading a bulleted list right now.
- **Definition lists** are handy for displaying terms followed by definitions or descriptions. For example, the dictionary is one huge definition list. In a definition list on a web page, your browser left-aligns the terms and indents the definitions underneath them.

In the following sections, you'll learn how to create all three types of list.

Ordered Lists

In an ordered list, HTML numbers each item consecutively, starting at some value (usually 1). The neat part about ordered lists in HTML is that you don't need to supply the numbers. Instead, the browser automatically adds the appropriate number next to each list item (sort of like the autonumber feature in Microsoft Word). This is handy for two reasons. First, it lets you insert and remove list items without screwing up your numbering. Second, HTML carefully aligns the numbers and list items, which isn't as easy if you do it on your own.

To create an ordered list, use ``, a block element (`` stands for "ordered list"). Then, inside the `` element, you place an `` element for each item in the list (`` stands for "list item").

For example, here's an ordered lists with three items:

```
<p>To wake up in the morning:</p>
<ol>
  <li>Rub eyes.</li>
  <li>Assume erect position.</li>
  <li>Turn on light.</li>
</ol>
```

In a browser, you'd see this:

To wake up in the morning:

1. Rub eyes.
2. Assume erect position.
3. Turn on light.

HTML inserts some space between the paragraph preceding the list and the list itself, as with all block elements. Next, it gives each list item a number.

Ordered lists get more interesting when you mix in the *start* and *type* attributes. The *start* attribute lets you start the list at a value other than 1. Here's an example that starts the counting at 5:

```
<p>To wake up in the morning:</p>
<ol start="5">
...
</ol>
```

This list will include the numbers 5, 6, and 7. Unfortunately, there's currently no way to count backward or to automatically continue counting from a previous list elsewhere on a page.

HTML doesn't limit you to numbers in your ordered list. The *type* attribute lets you choose the numbering style. You can use sequential letters and roman numerals, as described in Table 5-1. Figure 5-10 shows a few examples.

Table 5-1. Types of ordered lists.

Type Attribute	Description	Example
1	Numbers	1, 2, 3, 4...
a	Lowercase letters	a, b, c, d...
A	Uppercase letters	A, B, C, D...
i	Lowercase roman numerals	i, ii, iii, iv...
I	Uppercase roman numerals	I, II, III, IV...

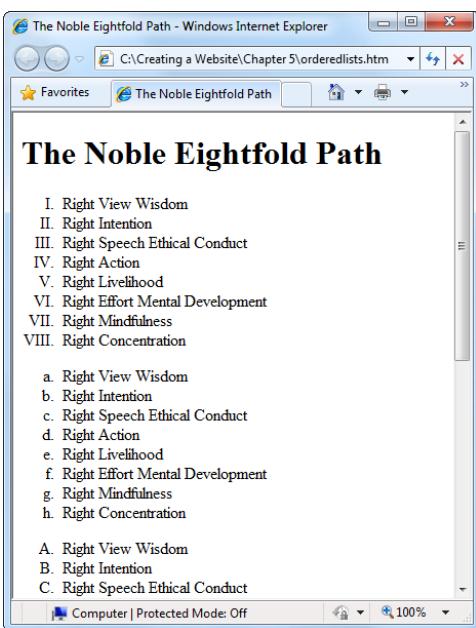


Figure 5-10:

The *type* attribute in action. For example, the code to start off the first list would be:

```
<ol type="I">
```

Unordered Lists

Unordered lists are similar to ordered lists except that they aren't consecutively numbered or lettered. The container element is ``, and you wrap each item inside an `` element. The browser indents each item in the list and automatically draws the bullets.

The most interesting frill that comes with unordered lists is the *type* attribute, which lets you change the style of bullet. You can use *disc* (a black dot, which is automatic), *circle* (an empty circle), or *square* (a filled-in square). Figure 5-11 shows the different styles.

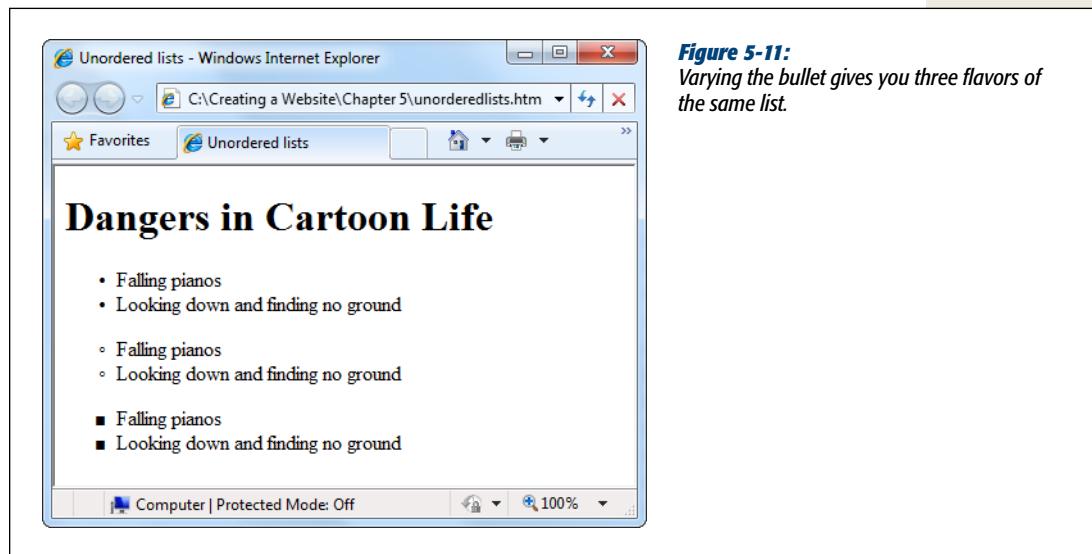


Figure 5-11:
Varying the bullet gives you three flavors of the same list.

Tip: Most web page editors have handy links so you can quickly create the different types of list. In Dreamweaver, look for the "ul" and "ol" icons in the Text tab of the Insert toolbar.

Definition Lists

Definition lists are perfect for creating your own online glossary. Each list item actually has two parts: a term (which the browser doesn't indent) and a definition (which the browser indents underneath the term).

Definition lists use a slightly different tagging system than ordered and unordered lists. First, you wrap the whole list in a dictionary list element (`<dl>`). Then you wrap each term in a `<dt>` element (dictionary term), and each definition in a `<dd>` element (dictionary definition). Here's an example:

```
<dl>
<dt>eat</dt>
<dd>To perform successively (and successfully) the functions of mastication, humectation, and deglutition.</dd>
<dt>eavesdrop</dt>
<dd>Secretly to overhear a catalogue of the crimes and vices of another or yourself.</dd>
<dt>economy</dt>
<dd>Purchasing the barrel of whiskey that you do not need for the price of the cow that you cannot afford.</dd>
</dl>
```

In a browser you'd see this:

eat

To perform successively (and successfully) the functions of mastication, humectation, and deglutition.

eavesdrop

Secretly to overhear a catalogue of the crimes and vices of another or yourself.

economy

Purchasing the barrel of whiskey that you do not need for the price of the cow that you cannot afford.

Nesting Lists

Lists work well on their own, but you can get even fancier by placing one complete list inside another. This technique is called *nesting* lists, and it lets you build multi-layered outlines and detailed sequences of instructions.

To nest a list, declare a new list inside an `` element in an existing list. For example, the following daily to-do list has three levels:

```
<ul>
<li>Monday
  <ol>
    <li>Plan schedule for week</li>
    <li>Complete Project X
      <ul style="square">
        <li>Preliminary Interview</li>
        <li>Wild Hypothesis</li>
        <li>Final Report</li>
      </ul>
    </li>
    <li>Abuse underlings</li>
  </ol>
</li>
```

```
<li>Tuesday
  <ol>
    <li>Revise schedule</li>
    <li>Procrastinate (time permitting). If necessary, put off
      procrastination until another day.</li>
  </ol>
</li>
<li>Wednesday
  ...
</ul>
```

Tip: When using nested lists, it's a good idea to use indents in your HTML document so you can see the different levels of list elements at a glance. Otherwise, you'll find it difficult to determine where each list item belongs.

In a nested list, the different list styles really start to become useful for distinguishing each level. Figure 5-12 shows the result of this example.

The screenshot shows a Mozilla Firefox browser window with the title bar "Nested List - Mozilla Firefox". The address bar displays "CaW/Chapter 5/nestedlist.htm". The main content area is titled "Nested List" and contains the heading "Weekly Schedule". Below the heading is a nested list structure:

- Monday
 - 1. Plan schedule for week
 - 2. Complete Project X
 - Preliminary Interview
 - Wild Hypothesis
 - Final Report
 - 3. Abuse underlings
- Tuesday
 - 1. Revise schedule
 - 2. Procrastinate (time permitting). If necessary, put off procrastination until another day.
- Wednesday
 - 1. Try to complete Monday tasks
 - 2. Write progress report. Consider ways to pad progress port.
 - "Intentionally left blank" sections
 - Font sizes above 20pt
 - Risqué pictures
- Thursday
 - 1. Determine why Project X isn't complete
 - 2. Research scapegoat possibilities
 - Boss
 - Co-workers
 - One-armed man
- Friday
 - 1. Defer tasks to next week
 - 2. Disrupt weekly meeting

Figure 5-12:

When you nest lists, browsers indent each subsequent list. Although you aren't limited in the number of levels you can use, you'll eventually run out of room in your browser window and force your text up against the right side of the page.

HTML Elements for Tables

An HTML table is a grid of cells built out of rows and columns. In the Bad Old Days of the Web, crafty web designers used invisible tables to line up pictures and arrange text into columns. Now style sheets fill that gap with top-notch layout features (as described in Chapter 9) and HTML tables are back to being just tables.

A Basic Table

You can whip up a table using these HTML elements:

- <table> wraps the whole shebang. It's the starting point for every table.
- <tr> represents a single table row. Every table element (<table>) contains a series of one or more <tr> elements.
- <td> represents a table cell (“td” stands for “table data”). For each cell you want in a row, you add one <td> element. You put the text (or numbers, or elements, or pretty much any HTML you like) that you want to appear in that cell inside the <td> element. If you put text in the cell, it displays in the same font as ordinary body text.
- <th> is an optional table element; you use it when you want to define a column heading. You can use a <th> element instead of a <td> element any time, although it usually makes the most sense in the first row of a table. Browsers format the text inside the <th> element in almost the same way as text in a <td> element, except that they automatically boldface and center the text (unless you apply different formatting rules with a style sheet).

Figure 5-13 shows a table at its simplest.

Here's a portion of the HTML used to create that table:

```
<table>
  <tr>
    <th>Rank</th>
    <th>Name</th>
    <th>Population</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Rome</td>
    <td>450,000</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Luoyang (Honan), China</td>
    <td>420,000</td>
  </tr>
  <tr>
```

```
<td>3</td>
<td>Seleucia (on the Tigris), Iraq</td>
<td>250,000</td>
</tr>
...
</table>
```

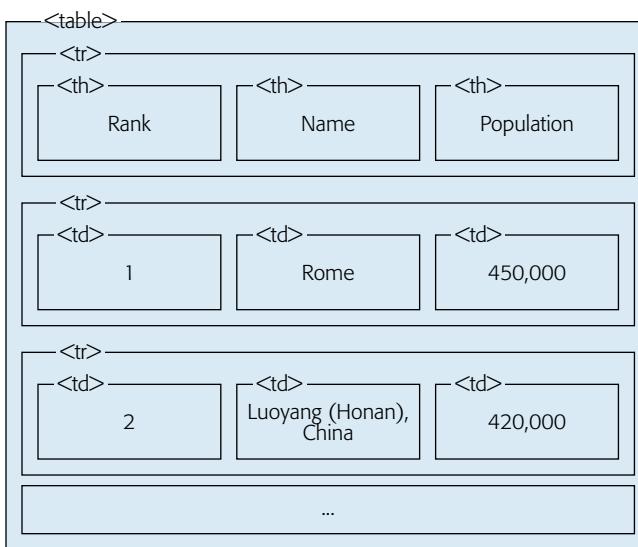
Largest Cities in the Year 100

Rank	Name	Population
1	Rome	450,000
2	Luoyang (Honan), China	420,000
3	Seleucia (on the Tigris), Iraq	250,000
4	Alexandria, Egypt	250,000
5	Antioch, Turkey	150,000
6	Anuradhapura, Sri Lanka	130,000
7	Peshawar, Pakistan	120,000
8	Carthage, Tunisia	100,000
9	Suzhou, China	n/a
10	Smyrna, Turkey	90,000

Figure 5-13:

Top: This basic table doesn't have any borders (which is standard), but you can still spot the signature sign that you're looking at a table: text lined up neatly in rows and columns.

Bottom: This behind-the-scenes look at the HTML powering the table above shows the <table>, <tr>, <th>, and <td> elements for the first three rows.



The markup for this table uses indented table elements to help you see the table's structure. Indenting table elements like this is always a good idea, as it helps you spot mismatched tags. In this example, the only content in the `<td>` elements is ordinary text. But you can put other HTML elements in cells, too, including images (the `` element).

Note: If you want to add table borders or control the sizing of table columns, you need to use a style sheet. Page 173 shows how to use style sheet properties to create table borders, and page 263 shows how to use them to control the width of table columns.

Cell Spans

Tables support *spanning*, a feature that lets a single cell stretch out over several columns or rows. Think of spanning as HTML's version of the Merge Cells feature in Microsoft Word and Excel.

Spanned cells let you tweak your tables in all kinds of funky ways. You can, for example, specify a *column span* to stretch a cell over two or more columns below it. Just add a `colspan` attribute to the `<td>` element you want to extend, and specify the total number of columns you want to skip.

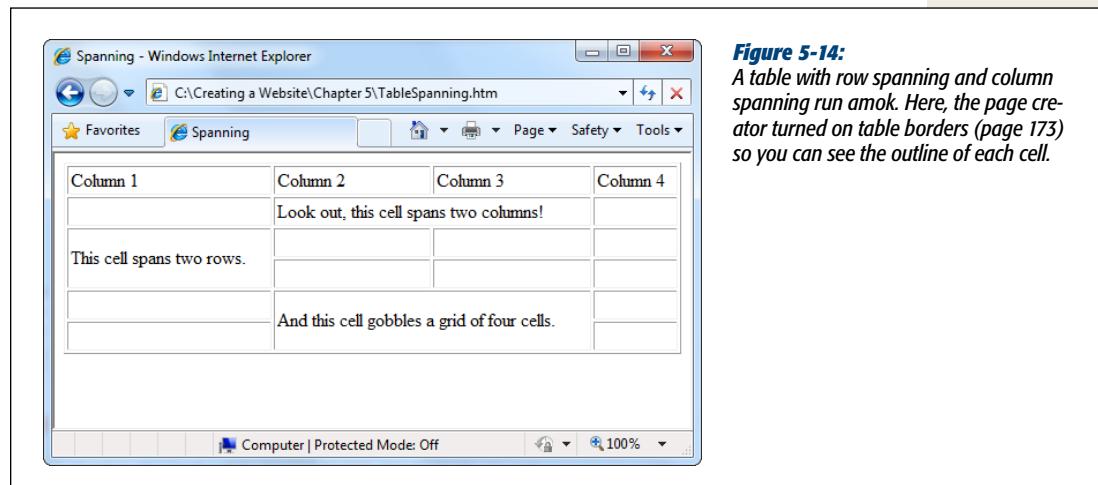
Here's an example that stretches a cell over two columns:

```
<table>
  <tr>
    <td>Column 1</td>
    <td>Column 2</td>
    <td>Column 3</td>
    <td>Column 4</td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td colspan="2">Look out, this cell spans two columns!</td>
    <td>&nbsp;</td>
  </tr>
  ...
</table>
```

Figure 5-14 shows this trick in action.

To make sure your table doesn't get mangled, you need to keep track of the total number of columns you have to work with in each row. In the previous example, the first row starts off by defining four basic columns:

```
<tr>
  <td>Column 1</td>
  <td>Column 2</td>
  <td>Column 3</td>
  <td>Column 4</td>
</tr>
```

**Figure 5-14:**

A table with row spanning and column spanning run amok. Here, the page creator turned on table borders (page 173) so you can see the outline of each cell.

In the next row, the second column extends over the third column, thanks to column spanning (see the markup below). As a result, the third `<td>` element actually represents the *fourth* column. That means you need only three `<td>` elements to fill up the full width of the table:

```
<tr>
  <!-- This fills column 1 -->
  <td>&nbsp;</td>

  <!-- This fills columns 2 and 3 -->
  <td colspan="2">Look out, this cell spans two columns!</td>

  <!-- This fills column 4 -->
  <td>&nbsp;</td>
</tr>
```

This same principle applies to *row spanning* and the `rowspan` attribute. In the following example, the first cell in the row leaks through to the second row:

```
<tr>
  <td rowspan="2">This cell spans two rows.</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
```

In the next row, the cell from above already occupies the first cell, so the first `<td>` element you declare actually applies to the second column. All in all, therefore, this row needs only three `<td>` elements:

```
<tr>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
</tr>
```

If you miscount and add too many cells to a row, you end up with an extra column at the end of your table.

Tip: Many Web page editors let you create spans by joining cells. In Dreamweaver and Expression Web, select a group of cells, right-click them, and then select Merge Cells.

Inline Formatting

As you learned earlier in this chapter, it's best not to format HTML too heavily. To get maximum control over your pages' appearance and to more easily update your website's look later on, you should head straight to style sheets (as described in the next chapter). However, a few basic formatting elements are truly useful. You're certain to come across them, and you'll probably want to use them in your own pages. These elements are all *inline* elements, so you use them inside a block element, like a paragraph, a heading, or a list.

Text Formatting: Italics and Bold

You know the elements for bold (``) and italic (`<i>`) formatting from Chapter 2. They're staples in the HTML world, letting you quickly format snippets of text. Here's an example:

```
<p>
<b>Stop!</b> The mattress label says <i>do not remove under penalty
of law</i> and you <i>don't</i> want to mess with mattress companies.
</p>
```

A browser displays that HTML like this:

Stop! The mattress label says *do not remove under penalty of law* and you *don't* want to mess with mattress companies.

To make life more interesting, HTML has a second set of elements that appear—at first glance—to do the same things. These are `` (for emphasized text) and ``.

Here's the previous example rewritten to use the `` and `` elements:

```
<p>
<strong>Stop!</strong> The mattress label says <em>do not remove under penalty
of law</em> and you <em>don't</em> want to mess with mattress companies.
</p>
```

Ordinarily, the `` element italicizes text, just like `<i>` does. Similarly, the `` element bolds text, just like `` does. The difference is philosophical.

Here's what the latest version of the HTML standard suggests:

- Use `` for text that has strong importance. The word “Stop!” in the previous example is a good example.

- Use `` for text that should be presented in bold but doesn't have greater importance than the rest of your text. This could include keywords, product names, and anything else that would be bold in print. For example, you might decide to bold the first three words of every paragraph in your document for stylistic reasons. It wouldn't be pretty, but `` would be a better choice than ``.
- Use `` for emphasized text, which would have a different inflection if read out loud. One example is the word "don't" in the previous example.
- Use `<i>` for text in an alternate voice, such as foreign words, technical terms, and anything else that you'd set in italics in print. The mattress warning in the previous example is a good model.

Note: Truthfully, there are very few web designers who haven't broken these guidelines at least a few times. The best advice is to be as consistent as possible in your own work. Also remember that you can change the way you emphasize your text long after you apply your markup. For example, you might decide that you don't want `` text to be bold, but to have a different color, a different font, or a different size. As you'll see in Chapter 6, you can change the formatting of any element with style sheets.

A Few More Formatting Elements

A few more elements can change the appearance of small snippets of text (see Figure 5-15), although you won't use them much, if at all. Do take the time to learn them, however, so you can be ready, in the rare case that you need one.

First up are two elements that change the size and placement of your text. You can use the `<sub>` element for *subscripts*—text that's smaller than and placed at the bottom of the current line. The `<sup>` element stands for *superscript*—small text at the top of the current line.

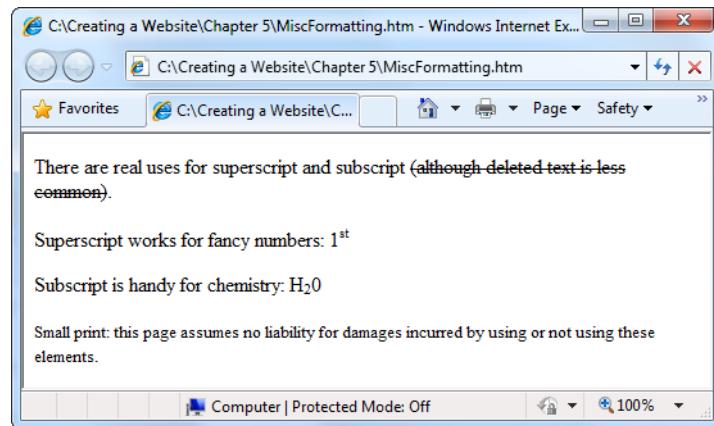
Next is the `` element, which is generally represents deleted text in a revised document or text that doesn't apply anymore. The browser displays this text, but crosses it out with a horizontal line. (HTML5 adds a complementary `<mark>` element for highlighting text, but it's still too new to get reliable browser support.)

Finally, the `<small>` element is the right way to format "small print"—for example, legalese at the bottom of a contract that a business is really hoping you'll overlook. In the past, the `<small>` element simply meant "small-looking text," and web designers rarely used it. HTML5, however, gave it a perfectly reasonable logical meaning, and with that a new lease on life.

Note: HTML also has a `<u>` element for underlining text, but webmasters consider it obsolete. If you really want to underline text, put it in a `` element (which applies no formatting on its own), and then use the `text-decoration` style property to format the span (see page 158).

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<p>There are real uses for superscript and subscript <del>(although deleted text is less common)</del>.</p>
<p>Superscript works for fancy numbers: 1<sup>st</sup></p>
<p>Subscript is handy for chemistry: H<sub>2</sub>O</p>
<p><small>Small print: this page assumes no liability for damages incurred by using or not using these elements.</small></p>
</body>
</html>
```

Figure 5-15:
Deleted text, superscript, subscript, and small print in action.



Special Characters

Not all characters are available from your keyboard. For example, what if you want to add a copyright symbol (©), a paragraph mark (¶), or an accented e (é)? Good news: HTML supports them all, along with about 250 relatives, including mathematical symbols and Icelandic letters. To add them, however, you need to use some sleight of hand. The trick is to use *HTML character entities*—special codes that browsers recognize as requests for unusual characters. Table 5-2 has some common options, with a sprinkling of accent characters.

Table 5-2. Common special characters.

Character	Name of Character	What to Type
©	Copyright	©
®	Registered trademark	®
¢	Cent sign	¢
£	Pound sterling	£
¥	Yen sign	¥
€	Euro sign	€ (but € is better supported)
°	Degree sign	°
±	Plus or minus	±
÷	Division sign	÷
×	Multiply sign	×
µ	Micro sign	µ
¼	Fraction one-fourth	¼
½	Fraction one-half	½
¾	Fraction three-fourths	¾
¶	Paragraph sign	¶
§	Section sign	§
«	Left angle quote, guillemot left	&lquo;
»	Right angle quote, guillemot right	&rquo;
¡	Inverted exclamation	¡
¿	Inverted question mark	¿
æ	Small ae diphthong (ligature)	æ
ç	Small c, cedilla	ç
è	Small e, grave accent	è
é	Small e, acute accent	é
ê	Small e, circumflex accent	ê
ë	Small e, dieresis or umlaut mark	ë
ö	Small o, dieresis or umlaut mark	ö
É	Capital E, acute accent	É

HTML character entities aren't just for non-English letters and exotic symbols. You also need them to deal with characters that have a special meaning according to the HTML standard—namely angle brackets (< >) and the ampersand (&). You shouldn't enter these characters directly into a web page because a browser will assume you're trying to give it a super-special instruction. Instead, you need to replace these characters with their equivalent character entity, as shown in Table 5-3.

Table 5-3. HTML character entities.

Character	Name of Character	What to Type
<	Left angle bracket	<
>	Right angle bracket	>
&	Ampersand	&
"	Double quotation mark	"

Strictly speaking, you don't need all these entities all of the time. For example, it's safe to insert ordinary quotation marks into your text by typing them in from your keyboard—just don't put them inside attribute names. Similarly, browsers are usually intelligent enough to handle an in-text ampersand (&) character appropriately, but it's better style to use the & code, so there's no chance a browser will confuse a naked ampersand with the beginning of a character entity. Finally, the character entities for in-text angle brackets are absolutely, utterly necessary.

Here's some flawed text that won't display correctly:

I love the greater than (>) and less than (<) symbols. Problem is,
when I type them my browser thinks I'm trying to use a tag.

And here's the corrected version, with HTML character entities. When a browser processes and displays this text, it replaces the entities with the characters you really want.

I love the greater than (>) and less than (<) symbols. Problem is,
when I type them my browser thinks I'm trying to use a tag.

Most web design tools insert the correct character entities as you type, as long as you're in Design view and not Code view.

Tip: To get a more comprehensive list of special characters and see how they look in your browser, check out www.webmonkey.com/reference/Special_Characters.

Non-English Languages

Although character entities work perfectly well, they can be a bit clumsy if you need to rely on them all the time. For example, consider the famous French phrase "We were given speech to hide our thoughts," shown here:

La parole nous a été donnée pour déguiser notre pensée.

Here's what it looks like with character entities replacing all the accented characters:

La parole nous a ét;é donnée pour déguiser notre
pensée.

French-speaking web creators would be unlikely to put up with this for long. Fortunately, there's a solution called *Unicode encoding*. Essentially, Unicode is a system that converts characters into the bytes that computers understand and can properly render in both HTML and the text that appears on your web page. By using Unicode encoding, you can create accented characters just as easily as if they were keys on your keyboard.

So how does it work? First, you need a way to get the accented characters into your web page. Here are some options:

- **Type it in.** Many non-English speakers will have the benefit of keyboards that include accented characters.
- **Use a utility.** In Windows, you can run a little utility called *charmap* (short for Character Map) that lets you pick from a range of special characters and copy your selected character to the clipboard so it's ready for pasting into another program. To run charmap, click the Start button and type *charmap* in the search box. Click on the program when it appears (Figure 5-16).
- **Use your web page editor.** Some web page editors include their own built-in symbol pickers. In Dreamweaver, you can use Insert→HTML→Special Characters→Other. In Expression Web, choose Insert→Symbol. Usually, this process inserts character entities, not Unicode characters. Though the end result is the same, your HTML markup will still include a clutter of codes.

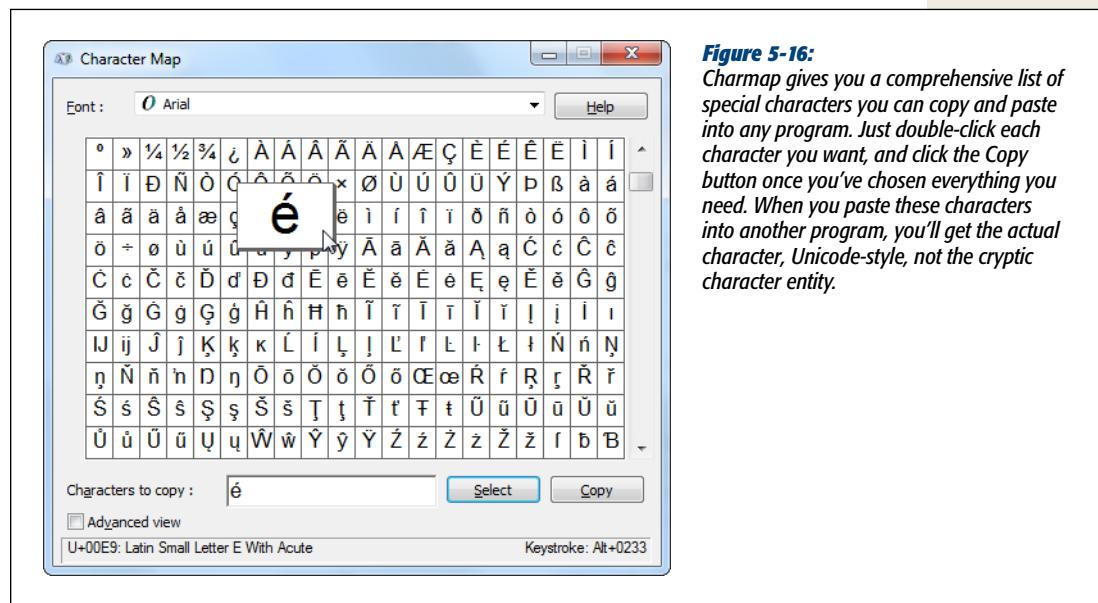


Figure 5-16:

Charmap gives you a comprehensive list of special characters you can copy and paste into any program. Just double-click each character you want, and click the Copy button once you've chosen everything you need. When you paste these characters into another program, you'll get the actual character, Unicode-style, not the cryptic character entity.

When you use Unicode encoding, make sure you save your web page correctly. This won't be a problem if you use a professional web page editor, which is smart enough to get it right the first time, but Unicode can trip up text editors. For example, in Windows Notepad, you need to choose File→Save As, and then pick UTF-8 from the Encoding list (see Figure 5-17). For the Mac'sTextEdit, select Format→Make Plain Text, go to Preferences→Open and Save→Plain Text File Encoding→Saving Files, and then select Unicode (UTF-8) from the drop-down list. Every time you re-save your file thereafter, Notepad and TextEdit will encode it correctly.

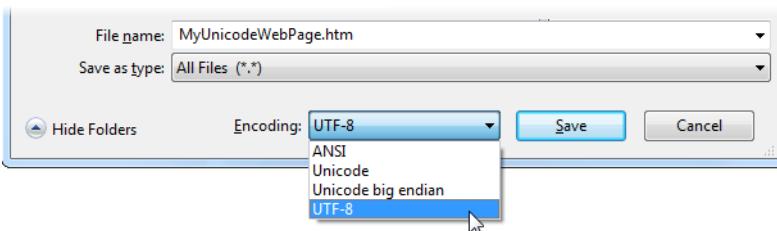


Figure 5-17:
The overwhelming standard of the Web, UTF-8 is a slimmed-down version of Unicode that saves space for normal characters. However, you need to explicitly tell Notepad to use UTF-8 encoding when you save a web page that includes special characters, like accented letters.

DESIGN TIME

Webifying Your Text

As you learned earlier in this chapter, text on the Web isn't like text in print. But sometimes it's hard to shake old habits. Here are some unwritten rules that can help you make good use of text in your web pages:

- **Split your text into small sections.** Web pages (and the viewers who read them) don't take kindly to long paragraphs.
- **Create short pages.** If a page goes on for more than two screens, split it into two pages. Not only does this make your pages easier to read, it gives you more web pages, which helps with the next point.
- **Divide your content into several pages.** The next step is to link these pages together (see Chapter 8). This gives readers the flexibility to choose what they want to read, and in what order.

- **Put your most important information on the first screen.** The basic idea is to make sure there's something eye-catching or interesting for visitors to read without having to scroll down. This technique is called designing *above the fold*. Well-designed newspapers use the same strategy; it gives newsstand visitors something interesting to read without having to flip over the folded broadsheet, hence the term "above the fold."
- **Proofread, proofread, proofread.** Typos and bad grammar shatter your site's veneer of professionalism and web-coolness.
- **Don't go wild with formatting until you understand style sheets.** If you break this rule, you'll leave a big mess that you'll only need to clean up later.

Style Sheets

Last chapter, you learned HTML's dirty little secret: It doesn't have much formatting muscle. If you want your web pages to look sharp, you need to add style sheets into the mix.

A style sheet is simply a document filled with formatting rules. Browsers read these rules and apply them when they display web pages. For example, a style sheet rule might say, "Make all the headings on this site bold and fuchsia, and draw a box around each one."

You want to put formatting instructions in a style sheet instead of embedding them in a web page for several reasons. The most obvious is *reuse*. For example, thanks to style sheets, you can create a single rule to, say, format level-3 headings, and every level-3 heading on every web page of your site will reflect that rule. The second reason is that style sheets help you write tidy, readable, and manageable HTML files. Because style sheets handle all your site's formatting, your HTML document doesn't need to. All it needs to do is organize your pages into logical sections. (For a recap of the difference between structuring and formatting a web page, see page 104.) Finally, style sheets give you more extensive formatting choices than HTML alone does. Using style sheets, you can control colors, borders, margins, alignment, and (to a limited degree) fonts.

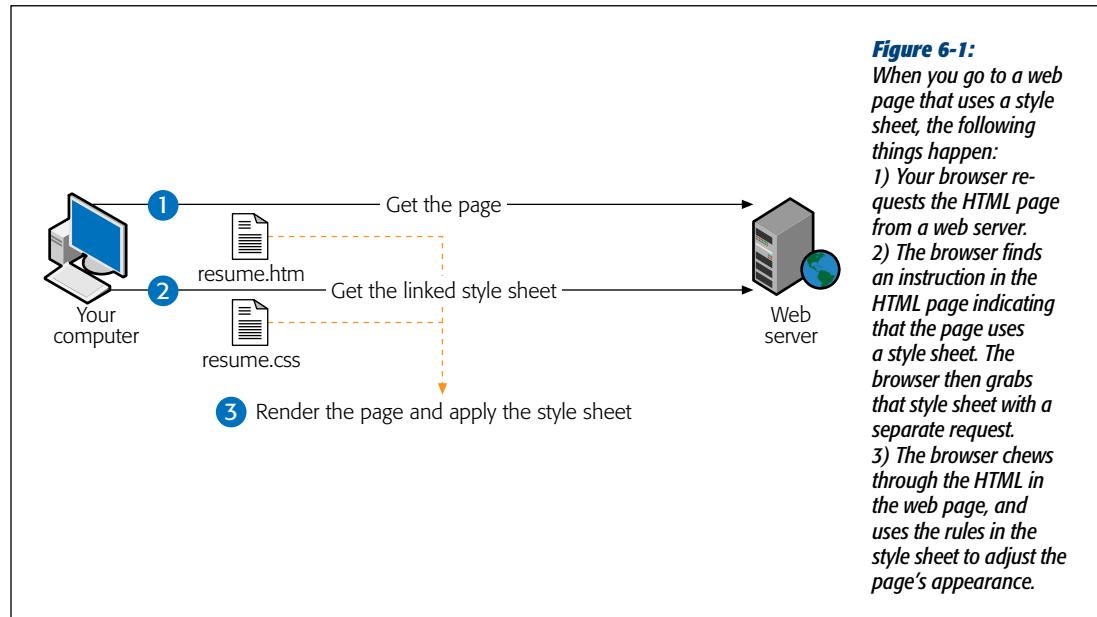
This chapter has two parts. First, you'll learn how style sheets work, and you'll see how to attach a style sheet to your web pages. Next, you'll explore the wide range of style sheet properties you can use to get the exact visual effects you want.

Note: In this chapter, you won't learn about every CSS property. For example, some properties apply primarily to pictures and layout; you'll learn about these properties in later chapters.

Style Sheet Basics

Style sheets use a standard that's officially known as CSS (Cascading Style Sheets). CSS is a system for defining style *rules*. These rules change the appearance of the elements in your web pages, tweaking details like color, font, size, borders, and placement.

When you use CSS in a web page, a browser reads both the page's HTML file and its style sheet rules. The browser then uses those rules to format the page. Figure 6-1 shows the process.



This system gives web weavers the best of both worlds—a rich way to format pages and a way to avoid mucking up your HTML beyond recognition. In an ideal world, the HTML document describes only the structure of your page (what's a header, what's a paragraph, what's a list, and so on), and the style sheet formats that page to give it its hot look.

The Three Types of Styles

Before you learn how to write CSS rules, you first have to think about *where* you're going to place those instructions. CSS gives you three ways to apply a style sheet to a web page:

- You store an **external style sheet** in a separate file. This is the most powerful approach, because it completely separates formatting rules from your HTML pages. It also gives you an easy way to apply the same rules to many pages.

- You embed an **internal style sheet** inside an HTML document (it goes right inside the <head> section of your page). You still have the benefit of separating the style information from the HTML, and if you really want, you can cut and paste the embedded style sheet from one page to another (although it gets difficult to keep all those copies synchronized if you make changes later on). You use an internal style sheet if you want to give someone a complete web page in a single file—for example, if you email someone your home page. You might also use an internal style sheet if you know that you aren’t going to use any of its style rules on another page.
- An **inline style** is a way to insert style sheet language directly inside the start tag of an HTML element. At first glance, this sounds suspicious. You already learned to avoid embedding formatting instructions inside a web page, because formatting details tend to be long and unwieldy. That’s true, but you might occasionally use the inline style approach to apply one-time formatting in a hurry. It’s not all that clean or structured, but it does work.

These choices give you the flexibility to either follow the CSS philosophy wholeheartedly (with external style sheets) or to use the occasional compromise (with internal style sheets or inline styles). Because style sheet language is always the same, even if you use a “lazier” approach like internal style sheets, you can always cut and paste your way to an external style sheet when you’re ready to get more structured.

UP TO SPEED

CSS Browser Compatibility

Before you embrace style sheets, you need to make sure they work on all the browsers your site visitors use. That’s not as easy to figure out as it should be, because there’s actually more than one version of the CSS standard—there’s the original CSS1, the slightly improved CSS2, the corrected CSS2.1, and the nearly ready CSS3. But the real problem is that browsers don’t necessarily support the entire CSS standard, no matter what version it is.

In this book you’ll focus on CSS1 and CSS2 properties known to be well-supported on all the major browsers. In a few rare cases, you’ll consider a feature that doesn’t yet have broad browser support (like the fancy font feature described on page 165). In this situation, the text will warn you of the compatibility risk and explain how to get the best, most consistent results. That said, don’t forget to test your pages in a variety of browsers to be sure they look right.

If you’re still concerned about whether a specific browser version supports a specific CSS feature, you may be interested in reading some of the web browser compatibility charts for CSS available on the Web. Two good resources are:

- **QuirksMode** (www.quirksmode.org/css/contents.html) includes piles of charts that painstakingly detail which browsers can handle which CSS features. Best of all, the charts are color-coded for easy reading and include multiple versions of popular browsers, like Firefox, Chrome, Safari, Opera, and Internet Explorer from version 5.5 to version 9.
- **When can I use** (<http://caniuse.com>) includes information about browser support for bleeding-edge features in CSS3, HTML5, and other related standards. You pick and choose which features you’re interested in, and the website generates the charts you need.

The Anatomy of a Rule

Style sheets contain just one thing: *rules*. Each rule is a formatting instruction that applies to a part of your page. A style sheet can contain a single rule, or it can hold dozens (or even hundreds) of them.

Here's a simple rule that tells a browser to display all `<h1>` headings in blue:

```
h1 { color: blue }
```

CSS rules don't look like anything you've seen in HTML markup, but you'll have no trouble with them once you realize that every rule uses only three ingredients: *selectors*, *properties*, and *values*. Here's the format that every rule follows:

```
selector { property: value }
```

And here's what each part means:

- The **selector** identifies the type of content you want to format. A browser then hunts down all the parts of a web page that match the selector. For now, you'll concentrate on selectors that match every occurrence of a specific page element, like a heading. Later in this chapter (page 147), you'll learn to create more sophisticated selectors that act on only specific sections of your page.
- The **property** identifies the type of formatting you want to apply. Here's where you choose whether you want to change colors, fonts, alignment, or something else.
- The **value** sets a value for the property defined above. This is where you bring it all home. For example, if your property is color, the value could be light blue or a queasy green.

Of course, it's rarely enough to format just one property of an HTML element. Usually, you want to format several properties at the same time. You can do this with style sheets by creating a rule like this:

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

This example changes the color of *and* centers the text inside an `<h1>` element. That's why style rules use the funny curly braces, { and }—so you can group as many formatting instructions inside them as you want. You separate one property from the next using a semicolon (;). It's up to you whether to include a semicolon at the end of the last property. Although it's not necessary, web-heads often do so to simplify adding additional properties onto the end of a rule.

Tip: As in an HTML file, CSS files let you use spacing and line breaks pretty much wherever you want. However, people often put each formatting instruction on a separate line (as in the example above) to make style sheets easy to read.

Conversely, you might want to create a single formatting instruction that affects several elements. For example, imagine you want to make sure that the first three heading levels, h1 to h3, all have blue letters. Rather than write three separate rules, you

can create a selector that includes all three elements, separated by commas. Here's an example:

```
h1, h2, h3 { color: blue }
```

Believe it or not, selectors, properties, and values are the essence of CSS. Once you understand these three ingredients, you're on your way to style sheet expertise.

Here are a few side effects of the style sheet system that you might not yet realize:

- A single rule can format a whole bunch of HTML. When you implement a rule for the kind of selectors listed above (called *type selectors*), that rule applies to every one of those elements. So when you specify blue h1 headings as in the example above, every h1 element in your page becomes blue.
- It's up to you to decide how much of your content you want to format. You can fine-tune every HTML element on your page, or you can write rules that affect only a single element, using the techniques discussed on page 147.
- You can create two different rules for the same element. For example, you could create a rule that changes the font of every heading level (`<h1>`, `<h2>`, `<h3>`, and so on), and then add another rule that changes the color of `<h1>` elements only. Just make sure you don't try to set the same property multiple times with conflicting values, or the results will be difficult to predict.
- Some elements have built-in style rules. For example, browsers always display text that appears in a `` element as boldfaced, even when the style sheet doesn't include a rule to do so. Similarly, browsers display text in an `<h1>` heading in a large font, with no style sheet rule necessary. But you can override any or all of these built-in rules using custom style rules. For example, you could explicitly set the font size of an `<h1>` heading so that it appears *smaller* than normal text. Similarly, you can take the underline off of a link, make the `` element italicize text instead of bolding it, and so on.

Don't worry about memorizing the kind of properties and values you can specify. Later in this chapter, after you see how style sheets work, you'll get acquainted with all the formatting instructions you can use.

Attaching a Style Sheet to a Page

Now it's time to see style sheets in action. Before you go any further, dig up the `resume.htm` file you worked on in Chapter 2 (it's also available from the Missing CD page at www.missingmanuals.com/cds/caw3). You'll use it to test out a new style sheet. Follow these steps:

1. Create the style sheet by opening a new file in any text editor, like Notepad or TextEdit.

Creating a style sheet is no different from creating an HTML page—it's all text. Many HTML editors have built-in support for style sheets (see the box on page 142 for more).

2. Type the following rule into your style sheet:

```
h1 { color: fuchsia }
```

This rule instructs your browser to display all `<h1>` elements in bright fuchsia lettering.

3. Save the style sheet with the name `resume.css`.

Like an HTML document, a style sheet can have just about any file name. As a matter of convention, however, style sheets almost always use the extension `.css`. For this example, make sure you save the style sheet in the same folder as your HTML page.

4. Next, open the `resume.htm` file.

If you don't have the `resume.htm` file handy, you can test this style sheet with any HTML file that has at least one `<h1>` element.

5. Add the `<link>` element to your HTML file.

The `<link>` element points your browser to the style sheet you wrote for your pages. You have to place the `<link>` element in the `<head>` section of your HTML page. Here's the revised `<head>` section of `resume.htm` with the `<link>` element added:

```
<head>
  <title>Hire Me!</title>
  <link rel="stylesheet" href="resume.css" />
</head>
```

The link element includes two details. The `rel` attribute indicates that the link points to a style sheet. The `href` attribute is the important bit; it identifies the location of your style sheet ("href" stands for "hypertext reference"). Assuming you put your style sheet in the same folder as your HTML file, all you need to supply is the file name. (If you put these two files in different folders, you need to specify the location of the `.css` file using the file path notation system you'll learn about on page 217.)

6. Save the HTML file, and then open it in a browser.

Here's what happens. Your browser begins processing the HTML document and finds the `<link>` element, which tells it to find an associated style sheet and apply all its rules. The browser then reads the first (and only, in this case) rule in the style sheet. To apply this rule, it starts by analyzing the selector, which targets all `<h1>` elements. Then it finds all the `<h1>` elements in the HTML page and applies the fuchsia formatting.

The style sheet in this example isn't terribly impressive. In fact, it probably seems like a lot of work to simply get a pink heading. However, once you get this basic model in place, you can quickly take advantage of it. For example, you could edit the style sheet to change the font of your `resume.htm` headings. Or, you could add rules to format other parts of the document. Simply save the new style sheet, and then refresh the web page to see the effect of the changed or added rules.

To see this at work, change the *resume.css* style sheet so that it has these rules:

```
body {
    font-family: Verdana,Arial,sans-serif;
    font-size: 83%;
}

h1 {
    border-style: double;
    color: fuchsia;
    text-align: center;
}

h2 {
    color: fuchsia;
    margin-bottom: 0px;
    font-size: 100%;
}

li {
    font-style: italic;
}

p {
    margin-top: 2px;
}
```

These rules change the font for the entire document (through the `<body>` element rule), tweak the two heading levels, italicize the list items, and shave off some of the spacing between paragraphs. Although you won't recognize all these rules at first, the overall model is the same as in the earlier resume example (content in the web page, formatting in the style sheet). Figure 6-2 shows the result.

Figure 6-2:
Left: By now, you can recognize a plain-vanilla web page.

Right: A style sheet revamps the entire page.

GEM IN THE ROUGH

Creating Style Sheets with Web Page Editors

Some web page editors, like Dreamweaver and Expression Web, have handy features for editing style sheets. To try them out, start by opening an existing style sheet or creating a new one. To create a style sheet in Dreamweaver, choose File→New, pick CSS in the Page Type list, and then click Create. To create a style sheet in Expression Web, choose File→New→CSS.

So far, you won't see anything to get excited about. But life gets interesting when you start to *edit* your style sheet. As you type, your web page editor pops up a list of possible style properties and values (see Figure 6-3). If you dig deeper, you'll find that both web editors have windows that let you build styles by pointing and clicking, as well as convenient shortcuts for applying styles to page elements.

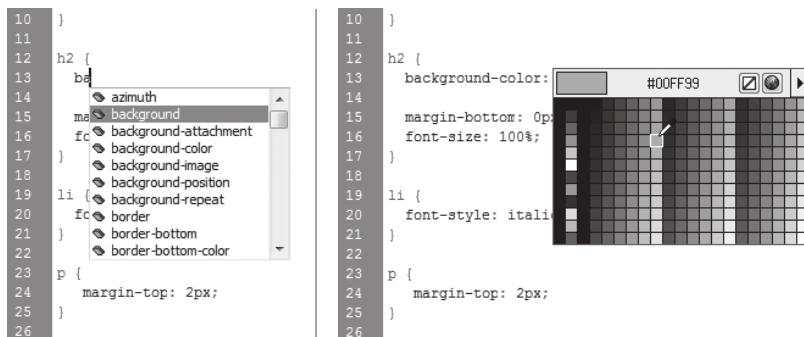


Figure 6-3:
As you edit a style sheet in Dreamweaver, it pops up lists of possible style properties (left) and property values. If you're dealing with colors, you even get this handy color picker (right), which translates the color codes in your style sheet into the actual color and displays the results. It's a great help for foggy memories, and saves more than a few keystrokes.

Using an Internal Style Sheet

The *resume.css* example demonstrates an external style sheet. External style sheets are everybody's favorite way to use CSS because they let you link a single lovingly crafted style sheet to as many web pages as you want. But there are times when you're not working on an entire website, and you'd be happy with a solution that's a little less ambitious.

An internal style sheet is one that, rather than being linked, you *embed* in the `<head>` area of your web page. Yes, it bulks up your pages and forces you to give each page a separate style sheet. But sometimes the convenience of having just one file that contains your page *and* its style rules makes this approach worthwhile.

To change the earlier example so that it uses an internal style sheet, remove the `<link>` element from your HTML markup and add the style rules in a `<style>` element inside the `<head>` section of the page, as shown here:

```
<head>
  <title>Hire Me!</title>
  <style>
    h1 { color: fuchsia }
  </style>
</head>
```

Using Inline Styles

If you want to avoid writing a style sheet altogether, you can use yet another approach. Inline styles let you insert the property and value portion of a style sheet rule right into the start tag for an HTML element. You don't need to specify the selector because browsers understand that you want to format only the element where you add the rule.

Here's how you use an inline style to format a single heading:

```
<h1 style="color: fuchsia">Hire Me!</h1>
```

The rule above affects only the `<h1>` element where you added it; any other `<h1>` headings in the page are unchanged.

Inline styles may seem appealing at first because they're clear and straightforward. You define the formatting information exactly where you want to use it. But if you try to format a whole page this way, you'll realize why web developers go easy on this technique. Quite simply, the average CSS formatting rule is long. If you need to put it alongside your content and copy it each time you use the element, you quickly end up with a web page that's mangled beyond all recognition. For example, consider a more complex heading that needs several style rules:

```
<h1 style="border-style: double; color: fuchsia; text-align: center">Hire
Me!</h1>
```

Even if this happens only once in a document, it's already becoming a loose and baggy monstrosity. So try to avoid inline styles if you can.

WORD TO THE WISE

Boosting Style Sheet Speed

External style sheets are a more efficient way to format websites than internal and inline styles because browsers use *caching*. Caching is a performance-improving technique in which browsers store a copy of some downloaded information on your computer so they don't need to download it again.

When a browser loads a web page that links to a style sheet, it makes a separate request for that style sheet, as shown back in Figure 6-1. However, if the browser loads another page that uses the *same* style sheet, it's intelligent enough to realize that it already has the right .css file on hand. As a result, it doesn't make the second request. Instead, it uses

the cached copy of the style sheet, which makes the page load a little faster. (Of course, browsers only cache things for so long. If you go to the same site tomorrow, the browser will have to re-request the style sheet.)

If you embed the style sheet in each of your web pages, the browser always downloads the full page with the duplicate copy of the style sheet. It has no way of knowing that you're using the same set of rules over and over again. Although this probably won't make a huge difference in page-download time, it could start to add up for a website with lots of pages. Speed is just one more reason web veterans prefer external style sheets.

The Cascade

By now, you might be wondering what the “cascading” part of “cascading style sheets” means. It refers to the way browsers decide which rules take precedence when you have multiple sets of rules.

For example, if an external style sheet indicates that `<h1>` headings should have blue letters, and then you apply bold formatting with an inline style, you'll end up with the sum of both changes: a blue-lettered, bold-faced heading. What happens if the rules conflict, however? For example, if one rule specifies blue text while another mandates red, which color setting wins?

To determine which rule has highest priority, consult the following list. It indicates the steps a browser follows when applying styles. The steps toward the bottom are the most powerful: The browser implements them after it applies the steps at the top, and they override any earlier formatting.

1. Browser's standard settings
2. External style sheet
3. Internal style sheet (inside the `<head>` element)
4. Inline style (inside an HTML element)

So, if an external style sheet conflicts with an internal style sheet, the internal style sheet wins.

Based on this, you might think that you can use this cascading behavior to your advantage by defining general rules in external style sheets, and then overriding them with the occasional exception using inline styles. In fact, you can, but there's a much

better option. Rather than format individual elements with inline style properties, you can use *class selectors* to format specific parts of a page (see page 147 for details), as you'll see later in this chapter.

Note: The “cascading” in cascading style sheets is a little misleading, because in most cases you won’t use more than one type of style sheet (for the simple reason that it can quickly get confusing). Most web artistes favor external style sheets primarily and exclusively.

Inheritance

Along with the idea of cascading styles, there’s another closely related concept—style *inheritance*. To understand style inheritance, you need to remember that in HTML documents, one element can contain other elements. Remember the unordered list element (``)? It contained list item elements (``). Similarly, a `<p>` paragraph element can contain character formatting elements like `` and `<i>`, and the `<body>` element contains the whole document.

Thanks to inheritance, when you apply formatting instructions to an element that contains *other* elements, that formatting rule applies to *every one of those other elements*. For example, if you set a `<body>` element to the font Verdana (as in the résumé style sheet shown earlier), every element inside that `<body>` element, including all the headings, paragraphs, lists, and so on, gets the Verdana font.

Note: Elements inherit most, but not all, style properties. For example, elements never inherit margin settings from another element. Look for the “Can Be Inherited” column in each table in this chapter to see whether CSS passes a property setting from one element to another through inheritance.

However, there’s a trick. Sometimes, formatting rules may overlap. In such a case, the most specific rule—that is, the one hierarchically closest to the element—wins. For example, settings you specify for an `<h1>` element will override settings you specified for a `<body>` element for all level-1 headings. Or consider this style sheet:

```
body {  
    color: black;  
    text-align: center;  
}  
  
ul {  
    color: fuschia;  
    font-style: italic;  
}  
  
li {  
    color: red;  
    font-weight: bold;  
}
```

These rules overlap. In a typical document (see Figure 6-4), you put an `` (list item) inside a list element like ``, which in turn exists inside the `<body>` element. In this case, the text for each item in the list will be red, because the `` rule overrides the `` and `<body>` rules that kick in first.

```

<!DOCTYPE html>
<html>
<head>
    <title>Inheriting styles</title>
    <link rel="stylesheet" href="Inheritingstyles.css" />
</head>
<body>
    <p>This is a test of style inheritance. List items are:</p>
    <ul>
        <li>Centered.</li>
        <li>Bold.</li>
        <li>Italicized.</li>
        <li>Displayed in red lettering.</li>
    </ul>
</body>
</html>

```

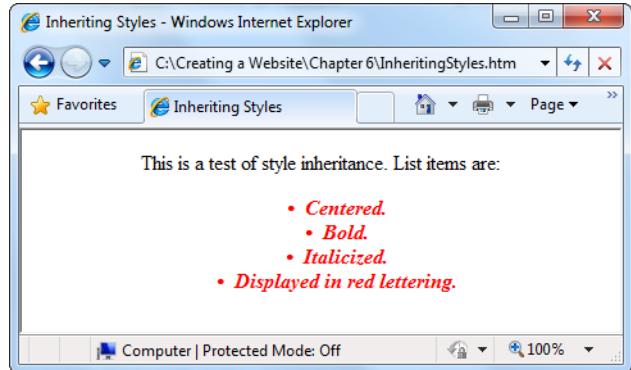


Figure 6-4:

When rules collide, the most specific element wins. In this example, your browser displays the list items in red because the rule for the `` element overrides the inherited properties from the `` and `<body>` elements (top). However, elements retain the style of an inherited rule if it doesn't conflict with another rule. In this example, that means the `` element gets *italics* and *center alignment* through inheritance (bottom).

Crafty style sheet designers can use this behavior to their advantage. For example, you might apply a font to the `<body>` element so that everything in your web page—headings, paragraph text, lists, and so on—has the same font. Then, you can judiciously override this font for a few elements by applying element-specific formatting rules.

Note: Although you probably won't see cascading styles in action very often, you'll almost certainly use style inheritance.

More Powerful Selectors

So far, you've seen style sheet rules that apply to every occurrence of a specific HTML element. The selectors in these universal styles are known as *type selectors*.

Type selectors are powerful, but not that flexible. Sometimes you need a way to modify just one section of your HTML document or even just a single element. You could use inline styles and put the formatting in the actual element tag itself, but that's messy. Fortunately, style sheets provide two practical solutions to this problem: class selectors and id selectors.

Class Selectors

Class selectors are one of the best style sheet tricks around. First, you single out specific elements in your page by giving them the same *class name*. Then, you tell your browser to apply formatting to all the elements that carry that class name.

To try this out, begin by choosing a descriptive class name. You can pick whatever name you want, as long as you stick to letters, digits, and dashes, and make sure that the first character is always a letter. The following example uses the class name FancyTitle.

Once you've chosen a class name, you need to define a rule for the class in your style sheet. This rule looks like any other, except that instead of using a tag name as the selector, you use the class name, preceded by a period (.).

```
.FancyTitle {  
    color: red;  
    font-weight: bolder;  
}
```

So how does a browser know when to apply a rule that uses a class selector? Browsers never apply class rules automatically; you have to add the class names to the elements you want to format. Here's an example that applies the FancyTitle class to a heading:

```
<h3 class="FancyTitle">Learning to Embroider</p>
```

As long as the class name in the element matches a class name in the style sheet, the browser applies the formatting. If the browser can't find a style associated with that class name, nothing happens.

Note: Class rules work *in addition* to any other rules. For example, if you create a rule for the <p> element, that rule applies to all paragraphs, including those that are part of a specialized class. However, if the class rule conflicts with any other rules, the class rule wins.

You can also create a rule that has a class name *and* specifies a type of element. For example, if you know that you only want to use the FancyTitle class with the <h3> element only, you would write the style rule like this:

```
h3.FancyTitle {  
    color: red;  
    font-weight: bolder;  
}
```

Most web designers use both element-specific class rules and more generic class rules (in other words, those that don't specify an element). Although you could stick exclusively with generic rules, if you know that you'll use a certain set of formatting options only with a specific type of element, it's good to clearly indicate this fact with an element-specific class rule. That way, you won't forget the purpose of the rule when you edit your style sheet later on.

id Selectors

Class selectors have a closely related cousin called *id selectors*. Like a class selector, the id selector lets you format just the elements you choose. And like a class selector, the id selector lets you pick a descriptive name. But instead of using a period, you use a number-sign character (#), as shown here:

```
#Menu {  
    border-width: 2px;  
    border-style: solid;  
}
```

Right now, you can apply this Menu rule to any element. However, you can also limit your id selector to a specific type of element by putting the element name before the number sign, like this:

```
div#Menu {  
    ...  
}
```

This example defines an id selector named Menu that can be applied only to <div> elements.

As with class rules, browsers don't apply id rules unless you specifically tell them to in your HTML. Instead of switching on the rules with a *class* attribute, however, you do so with the *id* attribute. For example, here's a <div> element that uses the Menu style:

```
<div id="Menu"
```

At this point, you're probably wondering why you would use an id selector—after all, the id selector seems almost exactly the same as a class selector. But there's one difference: You can assign a given id to just *one* element in a page. In the current example, that means you can label only one <div> with the Menu id. This restriction doesn't apply to class names, which you can reuse as many times as you like.

The id selector is a good choice if you want to format a single, never-repeated element on your page. The advantage here is that the id selector clearly indicates the special importance of that element. For example, if a page has an id selector named Menu or NavigationBar, the web designer automatically knows that the page features only one menu or navigation bar. Of course, you never *need* to use an id selector. Some web designers use class selectors for everything, whether the section is unique or not. It's really just a matter of personal preference.

Note: You'll learn about one more type of selector—contextual selectors—at the end of this chapter (page 180). But for now, it's time to shift focus. In the next sections, you'll capitalize on your hard-won style-sheet knowledge by learning about the dozens of formatting properties you can change.

Colors

It isn't difficult to inject some color into your web pages. Style sheet rules have two color-related properties, listed in Table 6-1. You'll learn about the types of values you can use when setting colors (color names, color codes, and RGB values) in the following sections.

Table 6-1. Color properties.

Property	Description	Common Values	Can Be Inherited?
color	The color of the text. This is a handy way to make headings or emphasized text stand out.	A color name, color code, or RGB color value.	Yes
background-color	The color behind the text for just that element.	A color name, color code, or RGB color value. You can also use the word "transparent."	No ¹

¹ The *background-color* style property doesn't use inheritance (page 145). If you give the `<body>` section of a page a blue background and you then place a heading on the page, the heading doesn't inherit the blue background. However, there's a trick. If you don't explicitly assign a background color to an element, its color is transparent. This means the color of the containing element shows through, which has the same effect as inheritance. So the heading in this example still ends up with the appearance of a blue background.

The *color* property is easy to understand; it's the color of your text. The *background-color* property is a little more unusual.

If you apply a background color to the `<body>` element of a web page, the whole page adopts that color, as you might expect. However, if you specify a background color for an individual *element*, like a heading, the results are a bit stranger. That's because CSS treats each element as though it were enclosed in an invisible rectangle. When you apply a background color to an element, CSS applies that color to just that rectangle.

For example, the following style sheet applies different background colors to the page, its headings, its paragraphs, and any bold text:

```
body {
  background-color: yellow;
}

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

```

p {
    background-color: lime;
}

b {
    background-color: white;
}

```

Figure 6-5 shows the result.

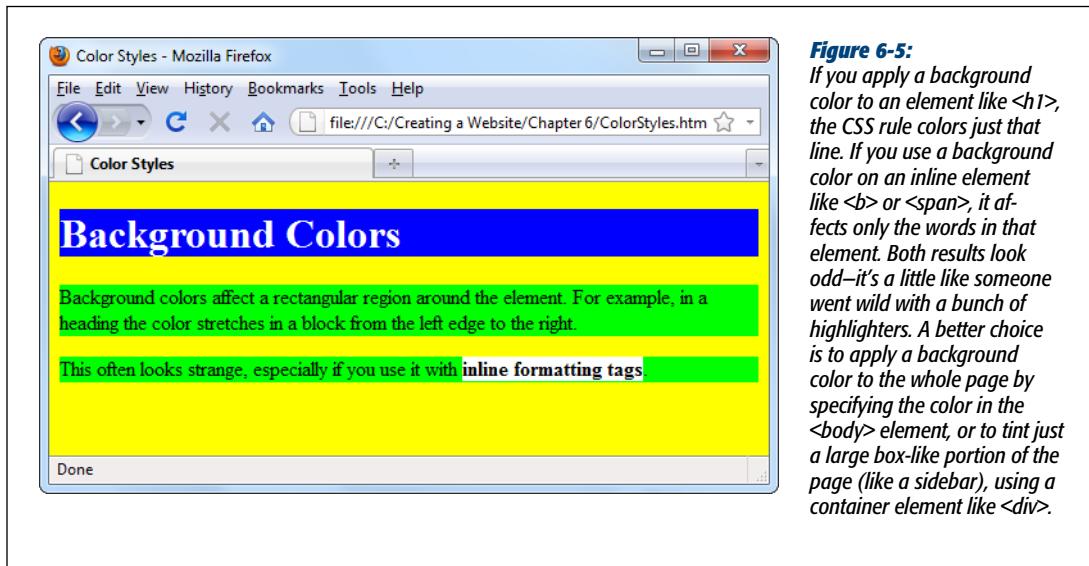


Figure 6-5:
If you apply a background color to an element like `<h1>`, the CSS rule colors just that line. If you use a background color on an inline element like `` or ``, it affects only the words in that element. Both results look odd—it's a little like someone went wild with a bunch of highlighters. A better choice is to apply a background color to the whole page by specifying the color in the `<body>` element, or to tint just a large box-like portion of the page (like a sidebar), using a container element like `<div>`.

Specifying a Color

The trick to using color is finding the code that indicates the exact shade of electric blue you love. You can go about this several ways. First, you can indicate your color choice with a plain English name (“lime”), as you’ve seen in the examples so far. Unfortunately, this system only works with a small set of 16 colors (aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow). Some browsers accept other names, but there’s no guarantee of cross-browser support for them, so it’s best to use another approach. CSS gives you two more options: hexadecimal color values and RGB (or *red-green-blue*) values.

Hexadecimal color values

With *hexadecimal color values* you use a strange-looking code that starts with a number sign (#). Technically, hexadecimal color values use three numbers to represent the amounts of red, green, and blue that go into creating a color. (You can create any color by combining various amounts of these three primary colors.) However, the hexadecimal color value combines these three ingredients into an arcane code that’s perfectly understandable to computers, but utterly baroque to normal people.

You'll find hexadecimal color notation kicking around the Web a lot, because it's the original format for specifying colors under HTML. However, it's about as intuitive as reading the 0s and 1s that power your computer.

Here's an example:

```
body {  
    background-color: #E0E0E0  
}
```

Even a computer nerd can't tell that #E0E0E0 applies a light gray background. To figure out the code for your favorite color, check out "Finding the Right Color" below.

RGB color values

The other approach to specifying color is RGB values. According to this more logical approach, you simply specify how much red, green, and blue you want to "mix in" to create your final color. Each component takes a number from 0 to 255. For example, a color composed of red, green, and blue, each set to 255, appears white; on the other hand, all those values set to 0 generates black.

Here's an example of a nice lime color:

```
body {  
    background-color: rgb(177,255,20)  
}
```

Finding the Right Color

Style sheets can handle absolutely any color you can imagine. But how do you find the color code for the perfect shade of sunset orange (or dead salmon) you need?

Sadly, there's no way this black-and-white book can show you your choices. But there *are* a lot of excellent color-picking programs online. For example, try www.colorpicker.com, where all you need to do is drag your mouse around a color gradient to preview the color you want (and to see its hexadecimal code). Or try www.colorschemer.com/online.html, which groups complementary colors together, which is especially helpful for creating websites that look professionally designed. If you use a web design tool like Dreamweaver or Expression Web, you have an even easier choice—the program's built-in color-picking smarts, as shown back in Figure 6-3.

Note: The RGB system lets you pick any of 16.7 million colors, which means that no color-picking website will show you every single possible RGB color code (if they do, make sure you don't hit the Print button; even with 10 colors per line, you'd wind up with thousands of pages). Instead, most sites limit you to a representative sampling of colors. This works, because many colors are so similar they're nearly impossible to distinguish.

The RGB color standard is also alive and well in many computer programs. For example, if you see a color you love in a professional graphics program like Photoshop (or even in a not-so-professional graphics program like Windows Paint), odds are there's a way to get the red, green, and blue values for that color. This gives you a great way to match the text in your web page with a color in a picture. Now that's a trick that will please even the strictest interior designer.

DESIGN TIME

Making Color Look Good

Nothing beats black text on a white background for creating crisp, clean, easy-to-read web pages with real presence. This black-and-white combination also works best for pages that have a lot of colorful pictures. It's no accident that almost every top website, from news sites (www.cnn.com) to search engines (www.google.com) to e-commerce shops (www.amazon.com) and auction houses (www.ebay.com), use the winning combination of black on white.

But what if you're just too colorful a person to leave your web page in plain black and white? The best advice is to follow the golden rule of color: *Use restraint*. Unless you're creating a sixties revival site or a Led Zeppelin tribute page, you don't want your pages to run wild with color. Here are some ways to inject a splash of color without letting it take over your web page:

- **Go monochrome.** That means use black, white, and one other dark color. Use the new color to emphasize an important design element, like subheadings in an article. For example, the *Time* magazine website uses its trademark red for some links and text.

- **Use lightly shaded backgrounds.** Sometimes, a faint wash of color in the background is all you need to perk up a site. For example, a gentle tan or gold can suggest elegance or sophistication (see the Harvard library site at <http://lib.harvard.edu>). Or light pinks and yellows can get shoppers ready to buy sleepwear and other feminine accoutrements at Victoria's Secret (www.victoriassecret.com).
- **Use color in a box.** Web designers frequently use shaded boxes to highlight important areas of a web page (check out this page at Wikipedia: <http://en.wikipedia.org>). You'll learn how to create boxes later in this chapter.
- **Be careful about using white text.** White text on a black or dark blue background can be striking—and strikingly hard to read. The rule of thumb is to avoid it unless you're trying to make your website seem futuristic, alternative, or gloomy. (Even if you do fall into one of these categories, you might get a stronger effect with a white background and a few well-chosen graphics with splashy electric colors.)

Text Alignment and Spacing

CSS includes a great many properties that let you control how text appears on a web page. If you ever wondered how to indent paragraphs, space out lines, or center a title, these are the tools you need.

Table 6-2 has the details on all your alignment options.

Table 6-2. Alignment and spacing properties.

Property	Description	Common Values	Can Be Inherited?
text-align	Lines up text on one or both edges of a page.	left, right, center, justify.	Yes
text-indent	Indents the first line of text (typically in a paragraph).	A pixel value (indicating the amount to indent) or percentage of the width of the containing element.	Yes
margin	Sets the spacing around the outside of a block element (page 107). To change the margin on just one side, use the similar properties margin-bottom, margin-left, margin-right, and margin-top.	A pixel value or percentage indicating the amount of space to add around the element.	No
padding	Sets the spacing around the inside of a block element. Has the same effect as “margin,” unless you have an element with a border or background color.	A pixel value or percentage indicating the amount of space to add around the element.	No
word-spacing	Sets the space between words.	A pixel value or percentage.	Yes
letter-spacing	Sets the space between letters.	A pixel value or percentage.	Yes
line-height	Sets the space between lines.	A pixel value or percentage. You can also use a multiple (for example, use 2 for double-spacing).	Yes
white-space	Tells the browser how to deal with spaces in your text.	normal, pre, nowrap.	Yes

For example, if you want to create a page with indented paragraphs (like a in novel or newspaper), use this style sheet rule:

```
p {
    text-indent: 20px
}
```

In the following sections, you’ll see examples that use the alignment and margin properties.

Alignment

Ordinarily, all the text on a web page lines up on the left side of the browser window. Using the *text-align* property, you can center that text, line it up on the right edge, or justify it. Figure 6-6 shows your options.

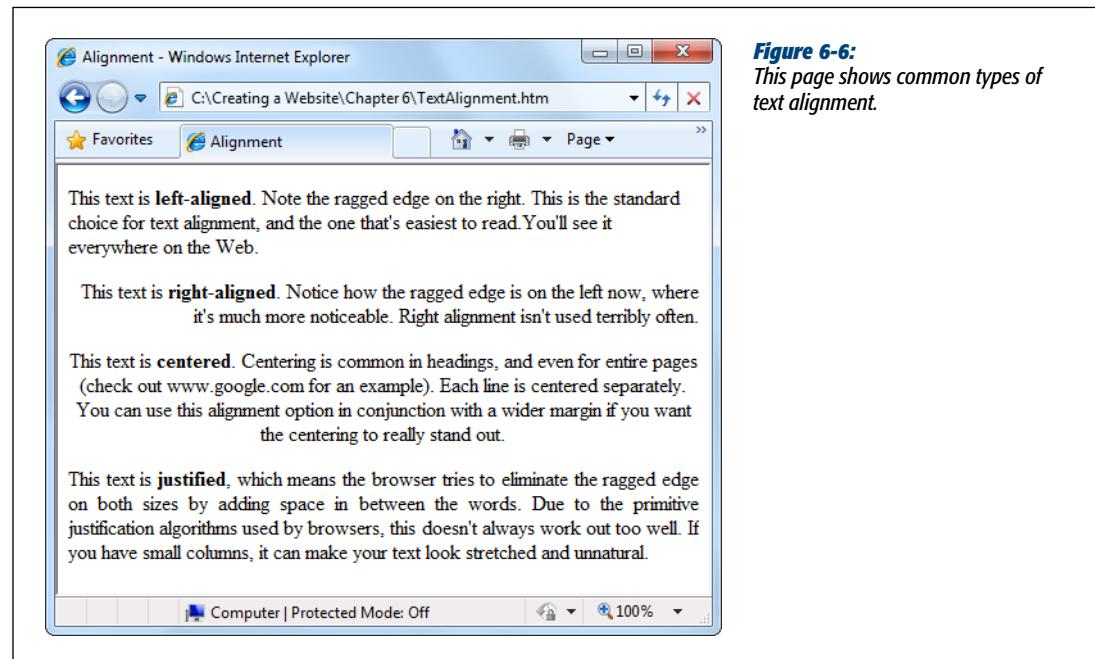
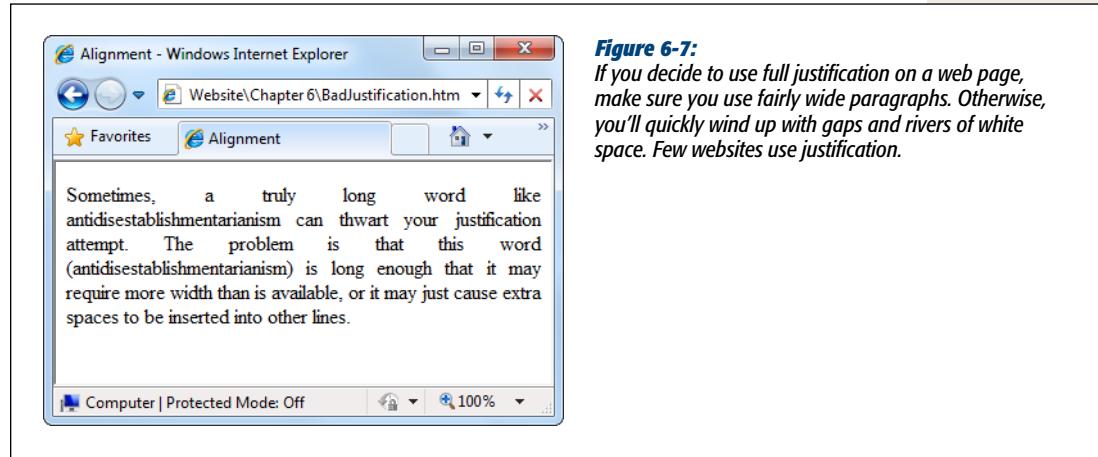


Figure 6-6:
This page shows common types of text alignment.

The most interesting alignment choice is full justification, which formats text so that it appears flush with both the left and right margins of a page, like the text in this book. You specify full justification with the *justify* setting. Originally, printers preferred full justification because it crams more words onto each page, reducing a book's page count and, therefore, its printing cost. These days, it's a way of life. Many people feel that text with full justification looks neater and cleaner than text with a ragged edge, even though tests show plain, unjustified text is easier to read.

Justification doesn't work as well in the web world as in print. A key problem is a lack of rules that split long words into syllables, hyphenate them, and extend them over two lines. Browsers use a relatively simplistic method to justify text. Essentially, they add words to a line one at a time, until no more words can fit, at which point they add extra spacing between the words to pad the line to its full length. By comparison, the best page layout systems for print analyze an entire paragraph and find the justification strategy that best satisfies every line. In problematic cases, a skilled typesetter may need to step in and adjust line breaks manually. Compared to this approach, web browsers are irredeemably primitive, as you can see in Figure 6-7.

**Figure 6-7:**

If you decide to use full justification on a web page, make sure you use fairly wide paragraphs. Otherwise, you'll quickly wind up with gaps and rivers of white space. Few websites use justification.

Spacing

To adjust the spacing around any element, use the *margin* property. For example, here's a rule that adds a fixed spacing of 8 pixels to all sides of a paragraph:

```
p {  
    margin: 8px;  
}
```

This particular rule doesn't have much effect, because 8 pixels is the margin web browsers apply around block elements on all sides to ensure a basic bit of breathing space. If you want to create dense pages of information, however, you might find this space allowance a bit too generous. Therefore, many website developers look for ways to slim down the margins a bit.

One common trick is to close the gap between headings and the text that follows them. Here's an example that puts this tightening into action using inline styles:

```
<h2 style="margin-bottom: 0px">This heading has no bottom margin</h2>  
<p style="margin-top: 0px">This paragraph has no top margin.</p>
```

You'll notice that this style rule uses the more targeted *margin-top* and *margin-bottom* properties to home in on just one margin at a time. You can use *margin-left* and *margin-right* to set side margins. Figure 6-8 compares some different margin choices.

Many CSS properties support a shorthand syntax that can compress several properties into one setting. For example, instead of setting your four margin properties separately (*margin-top*, *margin-bottom*, *margin-left*, and *margin-right*), you can set them at once with a rule like this:

```
p {  
    margin: 5px 10px 15px 20px;  
}
```

This sets the top margin to 5 pixels, the right margin to 10 pixels, the bottom margin to 15 pixels, and the left margin to 20 pixels. The key is to make sure you separate each number by a space—don’t be sloppy and add extra commas or semicolons.

If you’re daring, you can even use *negative* margins. Taken to its extreme, this can cause two elements to overlap. However, a better approach for overlapping elements is absolute positioning, a style trick you’ll pick up on page 245.

Note: Unlike most other CSS properties, elements never inherit margin settings. That means that if you change the margins of one element, other elements inside that element aren’t affected.

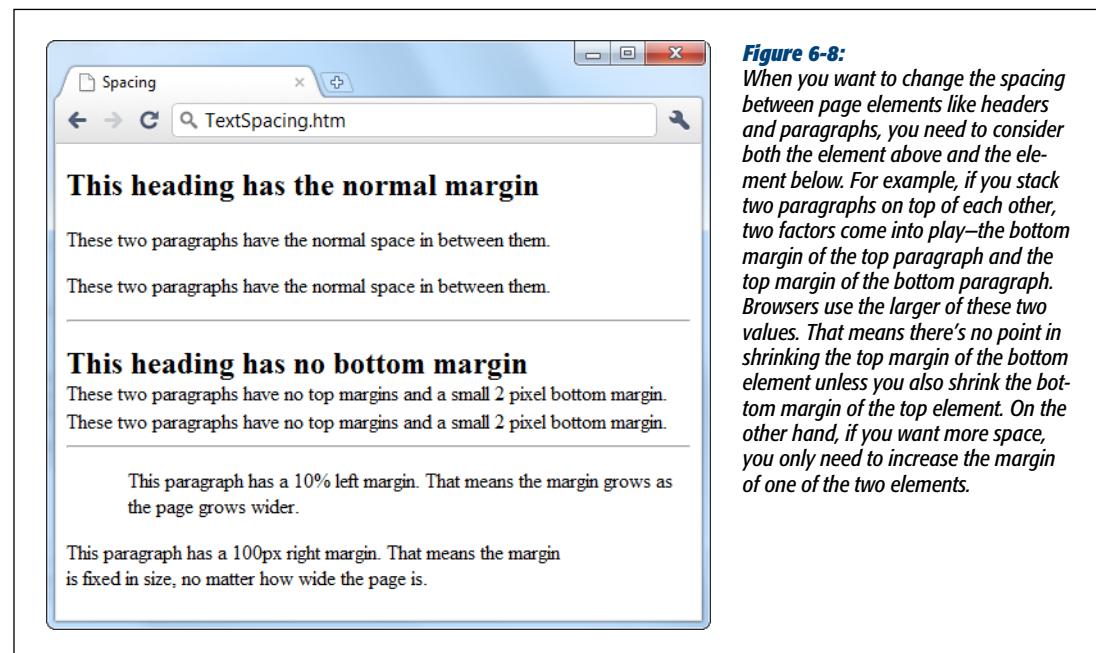


Figure 6-8:

When you want to change the spacing between page elements like headers and paragraphs, you need to consider both the element above and the element below. For example, if you stack two paragraphs on top of each other, two factors come into play—the bottom margin of the top paragraph and the top margin of the bottom paragraph. Browsers use the larger of these two values. That means there’s no point in shrinking the top margin of the bottom element unless you also shrink the bottom margin of the top element. On the other hand, if you want more space, you only need to increase the margin of one of the two elements.

White Space

As you learned in earlier chapters, HTML has a quirky way of dealing with spaces. If you put several blank spaces in a row, HTML treats the first one as a true space character but ignores the others. That makes it easy for you to write clear HTML markup, because you can add spaces wherever you like without worrying about it affecting your web page.

You already learned two ways to change how browsers deal with spaces: the character entity and the `<pre>` element. You can replace both of these workarounds with the *white-space* style sheet property.

First, consider the character entity. It serves two purposes—it lets you insert spaces that a browser won't ignore, and it prevents a browser from wrapping a line in the middle of a company name or some other important term. Here's an example of the latter technique:

```
<p>You can trust the discretion of  
Hush&nbsp;Hush&nbsp;Private&nbsp;Plumbers</p>
```

This works (the page displays the text "Hush Hush Private Plumbers" and doesn't wrap the company name to a second line), but it makes the markup hard to read. Here's the style-sheet equivalent with the *white-space* property set to *nowrap*:

```
<p>You can trust the discretion of  
<span style="white-space: nowrap">Hush Hush Private Plumbers</span></p>
```

To make this trick work, your HTML needs to wrap the company name in a container that applies the formatting. The `` element (page 118) is a good choice, because it doesn't apply any formatting except where you explicitly add it.

Now, consider the `<pre>` element, which tells a browser to pay attention to every space in the content inside it. On page 114, you saw how you could use `<pre>` to apply the correct spacing to an e. e. cummings poem. You can get the same effect by setting the *white-space* property of an element (say, a `<div>`, ``, or `<p>` element) to *pre*:

```
<p style="white-space: pre">Your browser won't ignore these  
      s p a c e s .</p>
```

When you use the *pre* value for the *white-space* property, the browser displays all spaces, tabs, and hard returns (the line breaks you create when you press the Enter key). But unlike the `<pre>` element, the *pre* value of the *white-space* property doesn't change the text font. If you want to use a fixed-width font like Courier to space your letters and spaces proportionally, you need to add a *font-family* property (see the next section).

Basic Fonts

Using the CSS font properties, you can choose a font family, font weight (its boldness setting), and font size (see Table 6-3). Be prepared, however, for a bit of web-style uncertainty, as this is one case where life isn't as easy as it seems.

Table 6-3. Font properties.

Property	Description	Common Values	Can Be Inherited?
font-family	A list of font names. The browser scans through the list until it finds a font that's on your visitor's computer. If it doesn't find a supported font, it uses the standard font the browser always uses.	A font name (like Verdana, Times, or Arial) or a generic font-family name: serif, sans-serif, monospace.	Yes
font-size	Sets the size of the font.	A specific size, or one of these values: xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger.	Yes
font-weight	Sets the weight of the font (how bold it appears).	normal, bold, bolder, lighter.	Yes
font-style	Lets you apply italic formatting.	normal, italic.	Yes
font-variant	Lets you apply small caps, which turns lowercase letters into smaller capitals (LIKE THIS).	normal, small-caps.	Yes
text-decoration	Applies a few miscellaneous text changes, like underlining and strikeout. Technically speaking, these aren't part of the font (the browser adds these).	none, underline, overline, line-through.	Yes
text-transform	Transforms text so that it's all capitals or all lowercase.	none, uppercase, lowercase.	Yes

Although most CSS font properties are straightforward, the *font-family* property has a nasty surprise—it doesn't always work. The inescapable problem you face is that no two computers have the same set of fonts installed, so the fonts you use to design your web page won't necessarily be the fonts on your visitors' machines. As a result, any font settings you specify are just recommendations. If a computer doesn't have the font you request, the browser reverts to the standard font it uses whenever it's on a site that doesn't have special font instructions.

Given that caveat, you probably wonder why you should bother configuring font choices at all. Well, here's one bit of good news. Instead of requesting a font and blindly hoping that it's available to a browser, you can create a list of *font preferences*. That way, the browser tries to match your first choice and, if it fails, your second choice, and so on. At the end of this list, you should specify one of the few standard fonts that almost all computers support. You'll see this technique at work in the next section.

If you absolutely *must* use an unusual font on your web page, you still have two options. For small bits of text (for example, headings or buttons), you can use an image,

as described in the box on page 204. For larger amounts of text, you could try out font embedding (described on page 165), but be warned that it's a next-generation feature that older browsers may not support.

Specifying a Font

To select a font, you use the *font-family* attribute. Here's an example that changes the font of an entire page:

```
body {
    font-family: Arial;
}
```

Arial is a *sans-serif* font found on just about every modern computer, including those running Windows, Mac OS, Unix, and Linux operating systems. (See Figure 6-9 for more about the difference between serif and sans-serif fonts.)

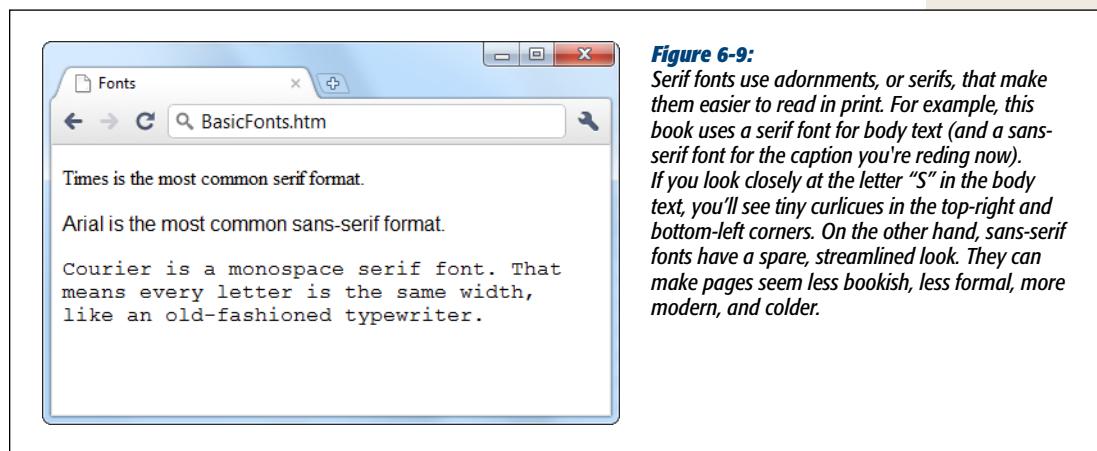


Figure 6-9:
Serif fonts use adornments, or serifs, that make them easier to read in print. For example, this book uses a serif font for body text (and a sans-serif font for the caption you're reading now). If you look closely at the letter "S" in the body text, you'll see tiny curlicues in the top-right and bottom-left corners. On the other hand, sans-serif fonts have a spare, streamlined look. They can make pages seem less bookish, less formal, more modern, and colder.

To be safe, when you create a font list, always end it with a generic font-family name. Every computer supports generic fonts under the *font-family* names *serif*, *sans-serif*, and *monospace*.

Here's the modified rule:

```
body {
    font-family: Arial, sans-serif;
}
```

At this point, you might be tempted to get a little creative with this rule by adding support for a less common sans-serif format. Here's an example:

```
body {
    font-family: Eras, Arial, sans-serif;
}
```

If Eras is relatively similar to Arial, this technique might not be a problem. But if the fonts are significantly different, it's a bad idea.

The first problem is that by using a nonstandard font, you're creating a page whose appearance may vary dramatically depending on the fonts installed on your visitor's computer. Whenever pages vary, it becomes more difficult to tweak them to perfection because you don't know exactly how they'll appear elsewhere. Different fonts take up different amounts of space, and if text grows or shrinks, the layout of other elements (like pictures) changes, too. Besides, is it really that pleasant to read Kidzz-FunScript or SnoopDawg font for long periods of time?

You'll see a more insidious problem if a visitor's computer has a font with the same name that looks completely different. Even worse, browsers may access an online database of fonts to try and find a similar font that's already installed on your guest's computer. This approach can quickly get ugly. At worst, either of these problems can lead to illegible text.

Tip: Most web page editors won't warn you when you apply a nonstandard font, so be on your guard. If your font isn't one of a small set of widely distributed web fonts (more on those in a moment), you shouldn't use it.

Finding the Right Font

To make sure your web page displays correctly, use a standard, widely available font. Just what are these standard fonts? Unfortunately, web experts aren't always in consensus.

If you want to be really conservative, you won't go wrong with any of these fonts:

- Times
- Arial
- Helvetica
- Courier

Of course, all of them are insanely boring. If you want to take more risk, you can use one of the following fonts, found on almost all Windows and Mac computers (but not necessarily on other operating systems, like Unix):

- Verdana
- Georgia
- Tahoma
- Comic Sans MS
- Arial Black
- Impact

If a font name has spaces or special characters, it's a good idea to wrap the whole thing in apostrophes or quotation marks. That means you should write:

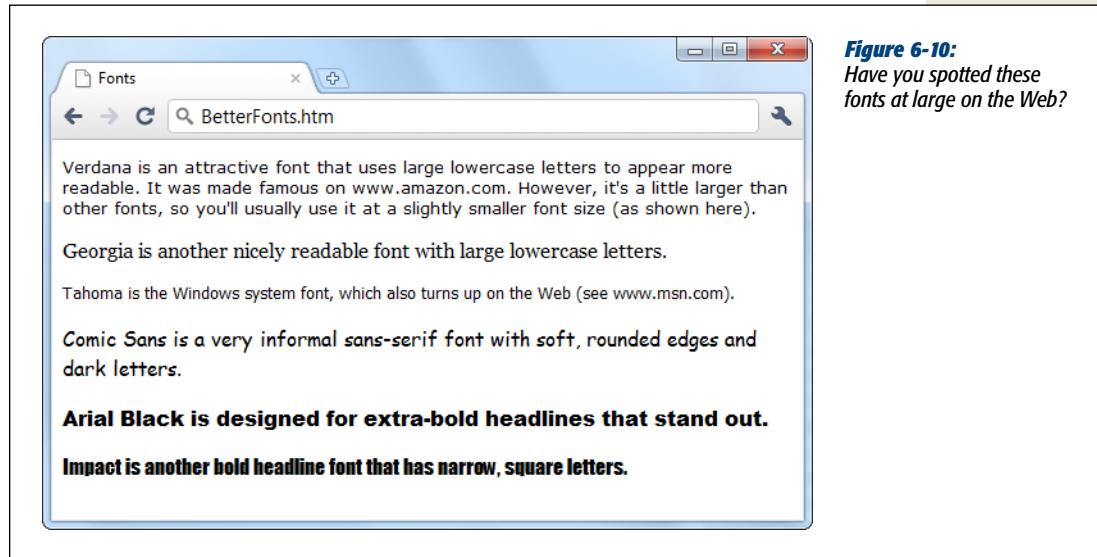
```
body {  
    font-family: "Comic Sans MS";  
}
```

rather than:

```
body {  
    font-family: Comic Sans MS;  
}
```

Most browsers won't care, but this practice helps avoid potential problems.

To compare these fonts, see Figure 6-10.



Verdana, Georgia, and Tahoma can all help give your web pages a more up-to-date look. However, the characters in Verdana and Tahoma start off a bit large, so you usually need to ratchet them down a notch in size (a technique described in the next section).

For a good discussion of fonts, the platforms that reliably support them, and the pros and cons of each font family (some fonts look nice onscreen, for example, but lousy when you print them out) see <http://tinyurl.com/cr9oyx> and <http://tinyurl.com/325f9qs>.

Font Sizes

Once you sort out the thorny issue of choosing a font, you may also want to change its size. It's important that you select a size that's readable and looks good. Resist the urge to shrink or enlarge text to suit your personal preferences. Instead, aim to match the standard text size you see on other popular websites.

Despite what you might expect, you don't have complete control over the size of the fonts on your web pages. Most site visitors use browsers that let them scale font sizes up or down, either to fit more text onscreen or, more commonly, to make text easy to read on a high-resolution monitor. In Internet Explorer and Firefox, you find these options in the View→Text Size menu.

A browser's font-size settings don't completely override the size you set in your web page, however. Instead, they tweak it up or down. For example, if you use a large font size on your web page (which corresponds to a setting of about 15 points in a word processor) and a visitor using Internet Explorer selects Page→Text Size→Larger, the text size grows about 20 percent, to 18 points.

The fact that your visitors have this kind of control is another reason you shouldn't use particularly small or large fonts on your pages. When you combine them with browser preferences, a size that's a little on the large side could become gargantuan, and text that's slightly small could turn unreadable. The best defense for these problems is to test your pages with different browsers *and* different text size preferences.

As you'll discover in the following sections, you can set font sizes several ways.

Keyword sizing

The simplest way to specify the size of your text is to use one of the size values listed in Table 6-2 above. For example, to create a really big heading and ridiculously small text, you can use these two rules:

```
body {  
    font-size: xx-small;  
}  
h1 {  
    font-size: xx-large;  
}
```

These size keywords are often called *absolute sizes*, because they apply an exact size to text. Exactly what size, you ask? Well, that's where it gets a bit complicated. These size details aren't set in stone; different browsers are free to interpret them in different ways. The basic rule of thumb is that the font size *medium* corresponds to a browser's standard text size (12 points), which is the size it uses if a website doesn't specify a text size. Every time you go up a level, you add about 20 percent in size. (For math geeks, that means that every time you go down a level, you lose about 17 percent.)

The standard font size for most browsers is 12 points (although text at this size typically appears smaller on Macs than on Windows PCs). That means *large* text measures approximately 15 points, *x-large* text is 18 points, and *xx-large* text is 27 points.

Figure 6-11 shows the basic sizes you can choose from.

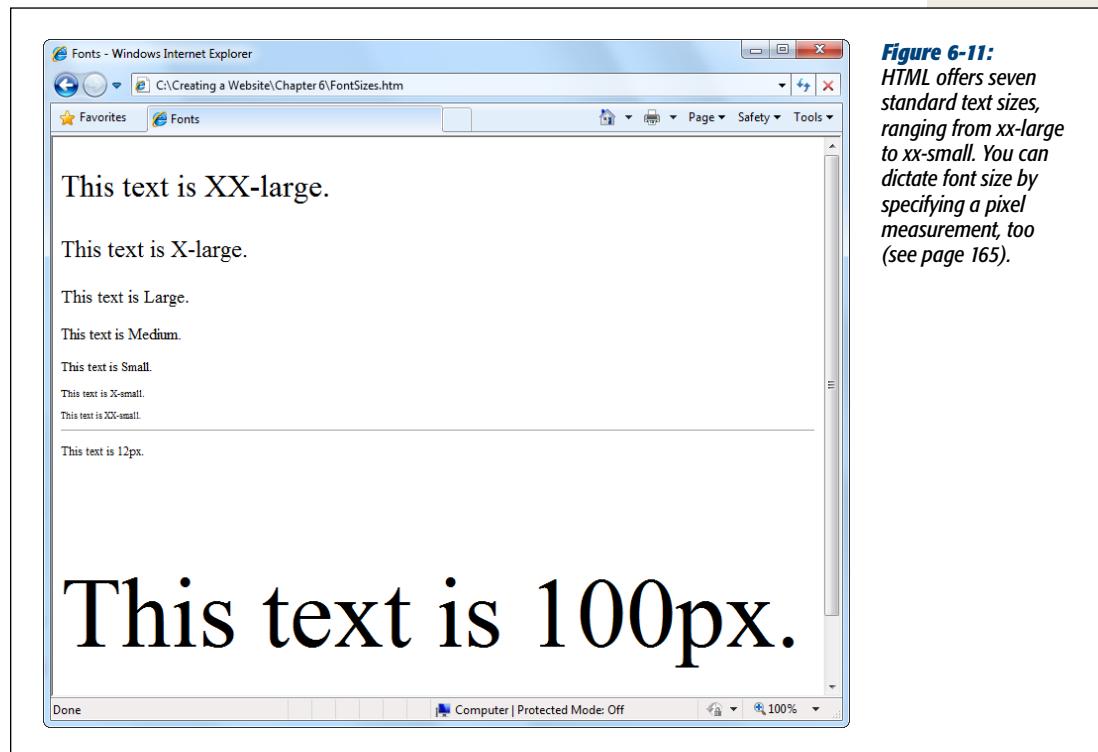


Figure 6-11:
HTML offers seven standard text sizes, ranging from xx-large to xx-small. You can dictate font size by specifying a pixel measurement, too (see page 165).

Note: When using size keywords, make sure your web page specifies a doctype. If you don't, Internet Explorer renders your page in the dreaded "quirks" mode, which makes your text one size larger than it should be. As a result, your page won't look the same in Internet Explorer as it does in other browsers, like Firefox.

Percentage sizing

Another *font-size* option is to use percentage size instead of size keywords. For example, if you want to make sure your text appears normal size, use this rule:

```
body {
    font-size: 100%;
}
```

And if you want to make your text smaller, use something like this:

```
body {
    font-family: Verdana, Arial, sans-serif;
    font-size: 83%;
}
```

It's just as easy to upsize text:

```
h1 {  
    font-size: 120%;  
}
```

But keep in mind that 100 percent always refers to the standard size of normal *paragraph* text, not the standard size of the element you’re styling, like an `<h1>` heading. So if you create a heading with text sized at 120 percent, your heading is going to be only a little bigger than normal paragraph text, which is actually quite a bit smaller than the normal size of an `<h1>` heading.

Using percentage sizes is the safest, most reliable way to size text. Not only does it provide consistent results across all browsers, it also works in conjunction with the browser size preferences described earlier.

Relative sizing

Another approach for setting font size is to use one of two *relative size* values—“larger” or “smaller.” This takes the current text size of an element and bumps it up or down a bit.

The easiest way to understand how this works is to consider the following style sheet, which has two rules:

```
body {  
    font-size: xx-small;  
}  
  
b {  
    font-size: larger;  
}
```

The first rule applies an absolute *xx-small* size to the whole page. If your page includes a `` element, the text inside that `` element *inherits* the *xx-small* size (see page 145 for a recap of inheritance), and then the second style rule steps the text up one notch, to *x-small*.

Now consider what happens if you edit the body text style above to use a larger font, like this:

```
body {  
    font-size: x-small;  
}  
  
b {  
    font-size: larger;  
}
```

Now all bold text will be one level up from *x-small*, which is *small*.

Relative sizes are a little trickier to get used to than absolute sizes. You’re most likely to use them if you have a style sheet with a lot of different sizes. For example, you might use a relative size for bold text if you want to make sure bold text is always a little bit bigger than the text around it. If you were to use an absolute size instead, the bold text would appear large in relation to small-sized paragraph text, but it wouldn’t stand out in a large-sized heading.

Tip: When you use absolute or relative sizes, you create flexible pages. If a visitor ratchets up the text size using his browser's preferences, the browser resizes all your other fonts proportionately.

Pixel sizing

Most of the time, you should rely on absolute and relative sizing for text. However, if you want to have more control, you can set a precise size for text by specifying its *pixel size*. Pixel sizes can range wildly, with 12 pixels and 14 pixels being about normal for body text. To specify a pixel size, use a number immediately followed by the letters *px*, as shown here:

```
body {  
    font-size: 11px;  
}  
h1 {  
    font-size: 24px;  
}
```

Note: Don't put a space between the number and the letters "px." If you do, your rule may work in Internet Explorer but it will thoroughly confuse other browsers.

As always, you need to test, refine, and retest your font choice to get the sizes right. Some fonts look bigger than others, so you should specify smaller sizes. Other fonts work well at larger sizes, but become less legible as you scale them down.

Internet Explorer treats pixel-sized copy differently from text that's sized any other way. If you use its text size feature (Page→Text Size), Internet Explorer adjusts all the text *except* for pixel-sized text. IE's zoom feature (Page→Zoom), on the other hand, affects everything on the page, including pictures and text. Other browsers don't give this special treatment to pixel-sized text. As a result, pixel-sized text can give you inconsistent results.

Embedded Fonts

For most of the Web's history, web designers had to live with the limited capabilities of the *font-family* style property. They learned to get the most out of the small set of standard fonts. But CSS3 introduces a feature called *@font-face*, which provides a way for browsers to download a web page's fonts on the fly (placing them in its temporary cache of pages and pictures), and use the fonts for just that page or website. As a result, web pages can use virtually any computer typeface.

Note: Technically, *@font-face* isn't new. It was a part of CSS 2, but dropped in CSS 2.1 when the browser makers couldn't cooperate. Now, in CSS3, there's a new drive to make *@font-face* a universal standard.

At first glance, `@font-face` seems like the perfect solution to the font-family woes web designers face. Unfortunately, you'll experience a certain amount of pain living on the bleeding edge of web design. The first challenge is getting support for your font format.

Web Formats for Fonts

Although all current browsers support `@font-face`, they don't all support the same type of font files. Internet Explorer, which has supported `@font-face` for years, supports only EOT (Embedded OpenType) font files, up until IE 9. The EOT format has a number of advantages—for example, it uses compression to reduce the size of the font file, and it allows strict website licensing so a font can't be stolen from one website and used on another. However, the .eot format never caught on, and no other browser uses it. Instead, other browsers have (until recently) stuck with the more familiar font standards used in desktop computer applications—that's TTF (TrueType) and OTF (OpenType PostScript). But the story's still not complete without using two more acronyms. Table 6-4 puts all the font formats in perspective.

Table 6-4. Embedded font formats.

Format	Description	Use With
TTF (TrueType) OTF (OpenType PostScript)	Your font will probably begin in one of these common desktop formats.	Firefox (before version 3.6), Chrome (before version 6), Safari, and Opera.
EOT (Embedded Open Type)	A Microsoft-specific format that never caught on with browsers except Internet Explorer.	Internet Explorer (before IE 9).
SVG (Scalable Vector Graphics)	An all-purpose graphics format you can use for fonts, with good but not great results (it's slower to display and produces lower-quality text).	Safari Mobile (on the iPhone and iPad before iOS 4.2 ²), and mobile devices using the Android operating system.
WOFF (Web Open Font Format)	The single format of the future, probably. Right now only the newest browsers support it.	Any browser that supports it, including Internet Explorer 9, Firefox 3.6, Chrome 6.

² iOS 4.2 was released in late 2010.

Note: For an even more detailed, up-to-date look at font support in different browsers, visit <http://tinyurl.com/cluxzz>.

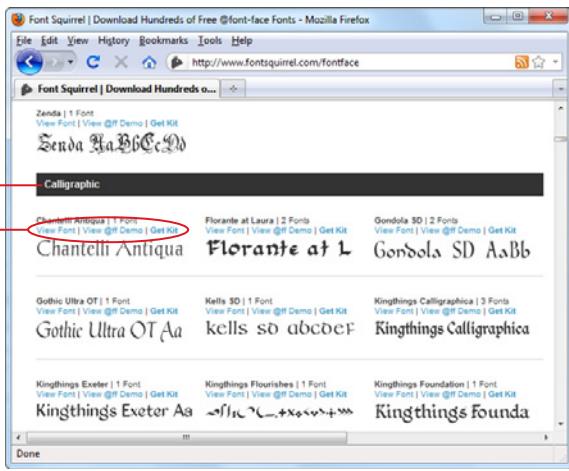
The final word is this: If you want to use the `@font-face` feature today, you need to support several standards at once. At a minimum, you need to supply your font in TTF or OTF format (either one is fine), the EOT format, and the SVG format. It's a good idea (but not essential) to also supply a WOFF font, which is likely to become

more popular and better supported in the future. And even if you follow the rules and supply all the required font formats, expect a few quirks; for example, people have reported that some browsers have trouble printing certain fonts.

Using a Font Kit

At this point, you're probably wondering where you can get the many font files you need. The easiest option is to download a ready-made font kit from the Web. That way, you get all the font files you need. The disadvantage is that your selection is limited to whatever you can find online.

One of the best places to find web font kits is the Font Squirrel website; you can see its hand-picked selection at www.fontsquirrel.com/fontface (see Figure 6-12).



The screenshot shows a Mozilla Firefox browser window displaying the Font Squirrel website. The page title is "Font Squirrel | Download Hundreds of Free @font-face Fonts - Mozilla Firefox". The main content area shows a section titled "Calligraphic" with several font families listed:

- Chantelli Antiqua** [1 Font] (circled in red)
- Florante at Laura** [2 Fonts]
- Gondola SD** [2 Fonts]
- Gothic Ultra OT** [1 Font]
- Hells SD** [1 Font]
- Kingthings Calligraphica**
- Kingthings Exeter Aa**
- Kingthings Exeter As**
- Kingthings Flourishes** [1 Font]
- Kingthings Foundation** [1 Font]
- Kingthings Founda**

Annotations with red arrows point to specific elements:

- A red arrow points to the "Calligraphic" category header with the text "Font category".
- A red arrow points to the "Chantelli Antiqua" entry with the text "Links to preview or download a font".

Figure 6-12:
Font Squirrel provides a few hundred high-quality fonts, organized into sections by type (Calligraphic, Grunge, and Retro, for example). Best of all, every font is free to use wherever you want—on your personal computer to create documents or on the Web to build web pages. When you find a font you like, click View Font to get a closer look at every letter, "View @ff Demo" to see the font in a sample web page via the @font-face feature, and Get Kit to download the font to your computer.

When you download a font kit, you get a compressed ZIP file that itself contains a number of files. For example, download the Chantelli Antiqua font shown in Figure 6-12, and you get these files:

```
Bernd Montag License.txt
Chantelli_Antiqua-webfont.eot
Chantelli_Antiqua-webfont.svg
Chantelli_Antiqua-webfont.ttf
Chantelli_Antiqua-webfont.woff
demo.html
stylesheet.css
```

The text file (*Bernd Montag License.txt*) provides licensing information that basically says you can use the font freely, but never sell it. The *Chantelli_Antiqua-webfont* files provide the font in four different file formats. (Depending on the font you pick,

you may get additional files for different variations of that font—for example, in bold, italic, and extra-dark styles.. Finally, the *stylesheet.css* file contains the style sheet rule you need to apply the font to your web page, and *demo.html* displays the font in a sample web page.

To use the Chantelli Antiqua font, you need to copy all the Chantelli_Antiqua-webfont files to the same folder as your web page. Then you need to register the font, so that it's available for use in your style sheet. To do that, you use a complex *@font-face* rule at the beginning of your style sheet, which looks like this (with the lines numbered for easy reference):

```
1  @font-face {  
2    font-family: 'ChantelliAntiquaRegular';  
3    src: url('Chantelli_Antiqua-webfont.eot');  
4    src: local('Chantelli Antiqua'),  
5         url('Chantelli_Antiqua-webfont.woff') format('woff'),  
6         url('Chantelli_Antiqua-webfont.ttf') format('truetype'),  
7         url('Chantelli_Antiqua-webfont.svg') format('svg');  
8  }
```

To understand what's going on in this rule, it helps to break it down line by line:

- Line 1: *@font-face* is the tool you use to officially register a font so you can use it elsewhere in your style sheet.
- Line 2: You can give the font any name you want. This is the name you'll use later, when you apply the font.
- Line 3: The first format you specify has to be the file name of the EOT file. That's because Internet Explorer gets confused by the rest of the rule and ignores the other formats. The *url()* function is a style sheet technique that tells a browser to download another file at the location you specify. If you put the font in the same folder as your web page, you can simply provide the file name here.
- Line 4: The next step is to use the *local()* function. This tells the browser the font name, and if that font just happens to be installed on the visitor's computer, the browser uses it. However, in rare cases this can cause a problem (for example, it could cause Mac OS X to show a security dialog box, depending on where your visitor has installed the font, or it could load a different font that has the same name). For these reasons, web designers sometimes use an obviously fake name to ensure that the browser finds no local font. One common choice is to use a meaningless symbol like *local('☺')*.
- Lines 5 to 7: The final step is to tell the browser about the other font files it can use. If you have a WOFF font file, suggest that first, as it offers the best quality. Next, tell the browser about the TTF or OTF file, and finally about the SVG file.

Tip: Of course, you don't need to type the *@font-face* rule in by hand (and you definitely don't need to understand all the technical underpinnings described above). You can simply copy the rule from the *stylesheet.css* file that's included in the web font kit.

Once you register an embedded font using the `@font-face` feature, you can use it in any style sheet. Simply use the familiar `font-family` property, and refer to the font family name you specified with `@font-face` (in line 2). Here's an example that leaves out the full `@font-face` details:

```
@font-face {  
    font-family: 'ChantelliAntiquaRegular';  
    ...  
}  
  
body {  
    font-family: 'ChantelliAntiquaRegular';  
}
```

This rule applies the font to the entire web page, although you could certainly restrict it to certain elements or use classes (page 147). Figure 6-13 shows a simple page that uses the Chantelli Antiqua font.

Note: You must register the font with `@font-face` before you use it in a style rule. Reverse the order of these two steps, and the font won't work properly.



The screenshot shows a Microsoft Internet Explorer window titled "Font Face Demo - Windows Internet Explorer". The address bar displays the local file path "C:\Creating a Website\Chapter 6\Embedded Fonts\Embeddedf". The page content is a resume with the following text:

Hire Me!
I am Lee Park. Hire me for your company, because my work is **off the hizzle**. As proof of my staggering computer skills and monumental work ethic, please enjoy this electronic resume.
Indispensable Skills
My skills include:
Fast typing (nearly 12 words/minute).
Extraordinary pencil sharpening.
Inventive excuse-making.
Negotiating with officers of the peace.
And I also know HTML!
Previous Work Experience
I have had a long and illustrious career in a variety of trades.
Here are some highlights:

The browser status bar at the bottom indicates "Computer | Protected Mode: Off" and "100%".

Figure 6-13:
This web page uses an embedded font to give its text a polished look that stands out from other sites.

Tip: Another good place to get free web fonts is the Google Font Directory at <http://code.google.com/webfonts>. You won't find nearly as many choices, but using them is easy. Instead of downloading the files and copying them to your website, you simply add a link to a Google style sheet that registers the font you want with `@font-face`. You can then use that font in your style sheet, without worrying about the specifics of font formats.

Using Your Own Fonts

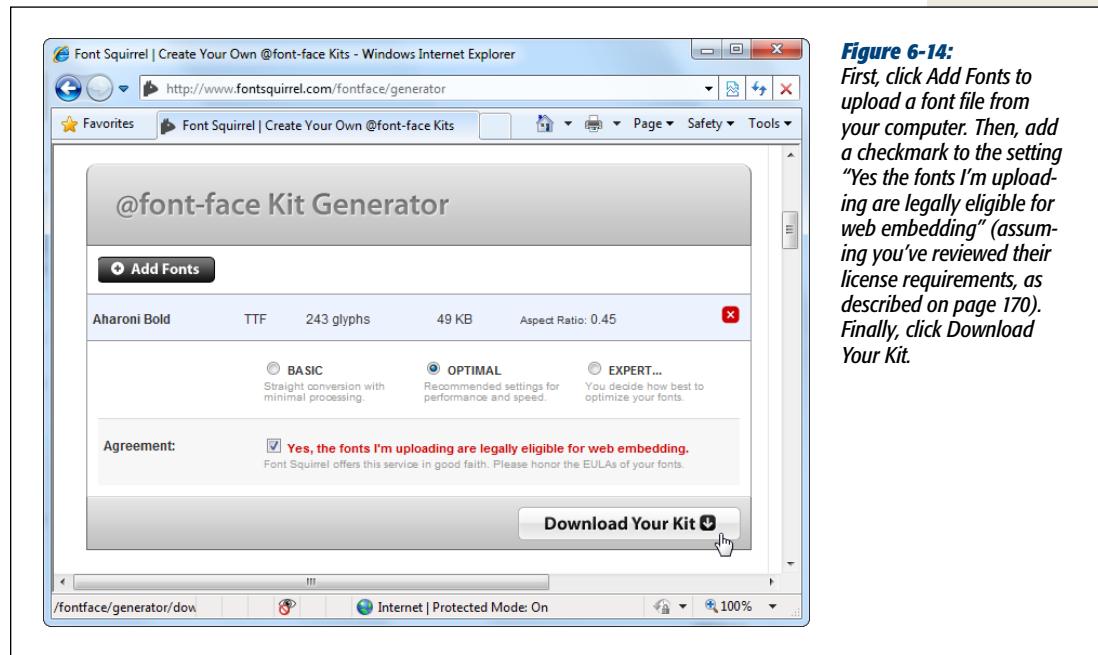
Font fanatics are notoriously picky about their typefaces. If you have a specific font in mind for your web pages, even the biggest free font library isn't enough. Fortunately, there's a fairly easy way to adapt any font for the Web. Using the right tool, you can take a TTF or OTF font file you already have, and create the other formats you need (EOT, SVG, and WOFF).

But before you take this road, it's important to get one issue out of the way. Ordinary fonts aren't free. That means it's not kosher to take a font you have on your computer and use it on your website, unless you have explicit permission from the font's creator. For example, Microsoft and Apple pay to include certain fonts with their operating systems and applications so you can use them to, say, create a newsletter in a word processor. However, this license doesn't allow you to put these fonts on a web server and use them in your pages.

Note: The cost of licensing a font varies greatly. Top-quality font foundries can charge hundreds of dollars in licensing charges, depending on the amount of traffic your website receives. Other font creators may let you use their fonts for a nominal fee or for free, provided you meet certain criteria (for example, you include some small-print note about the font you're using, or you have a noncommercial website that isn't out to make boatloads money). If you have a favorite font, the only way to know whether you need to pay for it is to contact the company or individual that made it. There's also a side benefit to reaching out. Skilled font makers often provide web-optimized versions of their creations, which look better than standard fonts when put online.

Once you know that you're allowed to use a specific font, you can convert it using a handy tool from Font Squirrel (the same website that offers the nifty free web font kits). To do so, surf to www.fontsquirrel.com/fontface/generator. Figure 6-14 shows you the three-step process you need to follow.

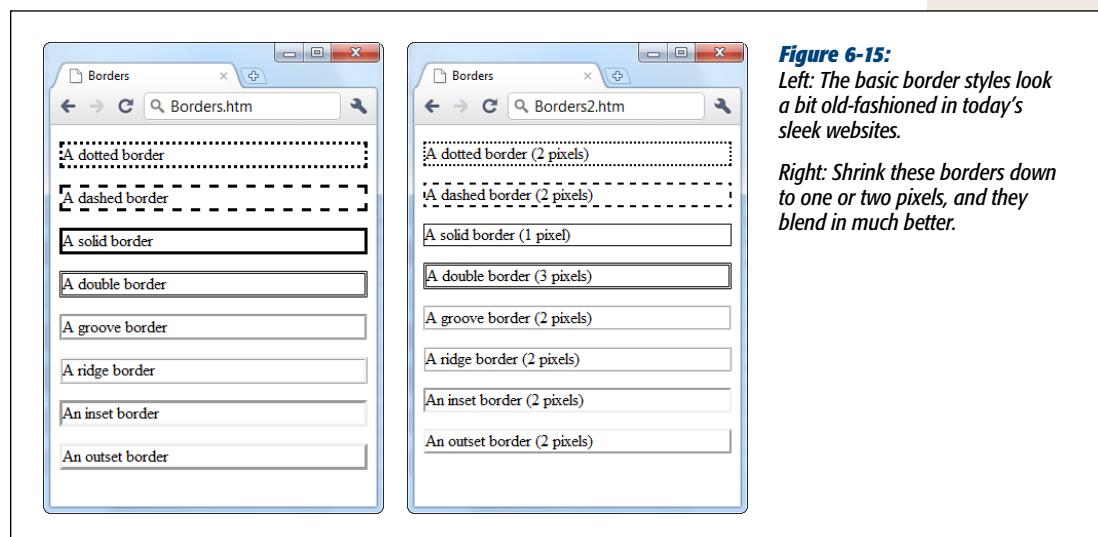
The web kit that Font Squirrel generates is just like the free ones described in the previous section. It even includes a style sheet that has the `@font-face` rule you need, along with a test web page.

**Figure 6-14:**

First, click Add Fonts to upload a font file from your computer. Then, add a checkmark to the setting "Yes the fonts I'm uploading are legally eligible for web embedding" (assuming you've reviewed their license requirements, as described on page 170). Finally, click Download Your Kit.

Borders

The last group of style sheet properties you'll learn about in this chapter lets you add borders to your web page (Figure 6-15). Borders are a great way to separate small pieces or entire blocks of content.

**Figure 6-15:**

Left: The basic border styles look a bit old-fashioned in today's sleek websites.

Right: Shrink these borders down to one or two pixels, and they blend in much better.

Table 6-5 lists the three key border properties.

Table 6-5. Border properties.

Property	Description	Common Values	Can Be Inherited?
border-width	Sets the thickness of the border line. Usually, you want to pare this down.	A pixel width.	No
border-style	Browsers have eight built-in border styles. The border style determines what the border looks like.	none, dotted, dashed, solid, double, groove, ridge, inset, outset.	No
border-color	The color of the border.	A color name, hexadecimal color code, or RGB value (see page 150).	No

Basic Borders

The first choice you make when you create a border is the style you want it to have. You can use a dashed or dotted line, a groove or a ridge, or just a normal thin hairline (which often looks best). Here's a rule that creates a dashed border:

```
p {
    border-style: dashed;
}
```

The standard border width is almost always too clunky. To make a border look respectable, reduce the border width to one or two pixels, depending on the border style:

```
p {
    border-style: dashed;
    border-width: 2px;
}
```

You can also use properties like *border-top-style* and *border-left-width* to set different styles, widths, and colors for every side of your element. Using many properties at once can occasionally create an unusual effect, but you usually don't need to get this detailed. Instead, check out the border optimization tips in the next section.

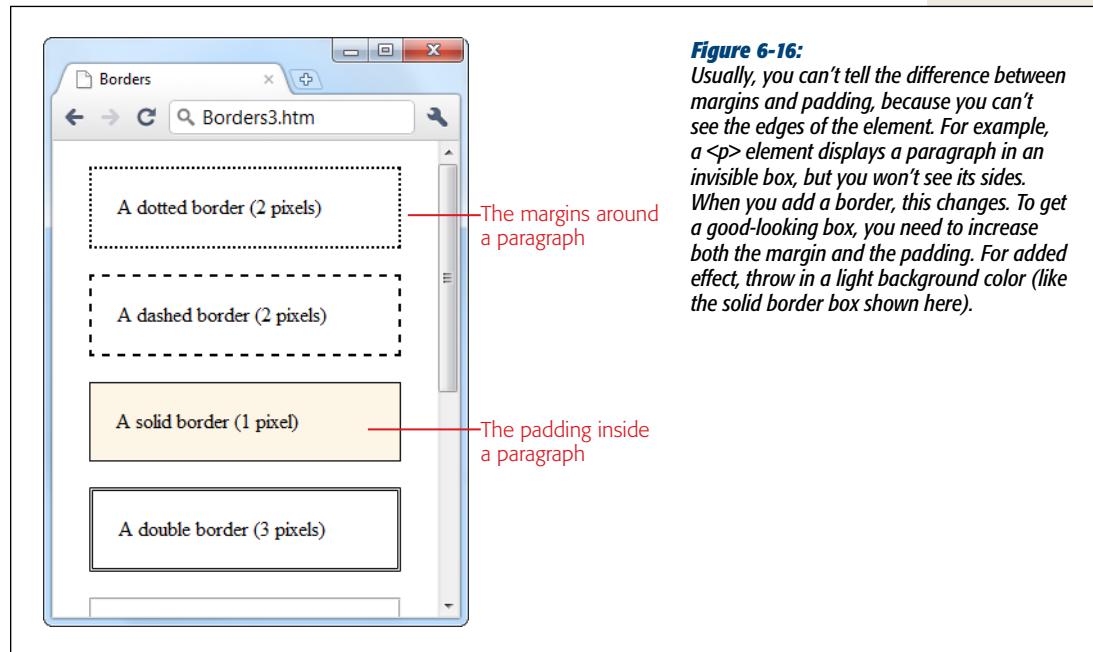
Making Better Borders

In Figure 6-15 the actual borders look fine, but they're too close to the text inside the boxes formed by the borderlines, as well as by the edges of the page.

To make a border stand out, consider using the *border* property in conjunction with three other properties:

- **background-color** (page 149) applies a background hue to your element. When used in conjunction with a border, it makes your element look like a floating box, much like a sidebar in a magazine.

- **margin** (page 153) lets you set the spacing between your border box and the rest of your page. Increase the margin so that your boxes aren't crowded up against the rest of the page's content or the sides of a browser window.
- **padding** works like the *margin* property, but it sets spacing *inside* your element, between the edges of the box and the actual content within it. Increase the padding so that there's a good amount of space between a border and your box text. Figure 6-16 shows the difference between margin and padding.

**Figure 6-16:**

Usually, you can't tell the difference between margins and padding, because you can't see the edges of the element. For example, a `<p>` element displays a paragraph in an invisible box, but you won't see its sides. When you add a border, this changes. To get a good-looking box, you need to increase both the margin and the padding. For added effect, throw in a light background color (like the solid border box shown here).

Here's an example of a paragraph that looks like a shaded box:

```
p {
  background-color: #FDF5E6;
  margin: 20px;
  padding: 20px;
  border-style: solid;
  border-width: 1px;
}
```

Figure 6-16 shows how the *margin*, *padding*, and *background-color* properties change an ordinary paragraph into a shaded box.

Using Borders with Tables

As you learned in Chapter 5 (page 124), tables start out as a borderless collection of cells. But using a style sheet, you can easily outfit your table with custom borders. You simply need to apply the border properties to some combination of the `<tr>`, `<td>`, `<th>`, and `<table>` elements.

For example, the following style sheet rules set a thin blue border around every cell, and a thick blue border around the table itself:

```
table {
    border-width: 3px;
    border-style: solid;
    border-color: blue;
}
td, th {
    border-width: 1px;
    border-style: solid;
    border-color: blue;
}
```

Figure 6-17 shows the result.

Figure 6-17:

Compare a standard HTML table with no border (top) to the same table with custom border added using style rules (bottom).

Rank	Name	Population
1	Rome	450,000
2	Luoyang (Honan), China	420,000
3	Seleucia (on the Tigris), Iraq	250,000
4	Alexandria, Egypt	250,000
5	Antioch, Turkey	150,000
6	Amuradhapura, Sri Lanka	130,000
7	Peshawar, Pakistan	120,000
8	Carthage, Tunisia	100,000
9	Suzhou, China	n/a
10	Smyrna, Turkey	90,000

Rank	Name	Population
1	Rome	450,000
2	Luoyang (Honan), China	420,000
3	Seleucia (on the Tigris), Iraq	250,000
4	Alexandria, Egypt	250,000
5	Antioch, Turkey	150,000
6	Amuradhapura, Sri Lanka	130,000
7	Peshawar, Pakistan	120,000
8	Carthage, Tunisia	100,000
9	Suzhou, China	n/a
10	Smyrna, Turkey	90,000

Tutorial: Building a Style Sheet

You've now explored a huge range of formats you can apply to web pages, from psychedelic colors to fancy fonts. Even though you know what style sheets can do, you're probably less sure about how to build a practical style sheet—one that condenses complex formatting down to a simple set of rules, but remains flexible enough to grow with your website.

That's OK. Creating style sheets is an art and takes a fair bit of practice. The following tutorial will help get your mind in gear.

Creating a Well-Structured Style Sheet

Before you can write any style sheet rules, you need to think hard about your web page and how you structured it. For example, imagine your website includes the page of book reviews shown in Figure 6-18.



The screenshot shows a Mozilla Firefox window with the title bar "The Pessimist - Mozilla Firefox". The address bar displays "file:///C:/Creating a Website/Chapter 6/PessimistReviews.htm". The main content area is titled "The Pessimist's Review Site". It contains several sections of text, each starting with a bold heading and followed by descriptive text. The headings include "How To Lose Friends and Fail in Life" and "Europe 2011: Great Places to Miss". The descriptive text for the first section is attributed to "Chris Chu" and the second to "Antonio Cervantes". The overall layout is a single column of text.

Figure 6-18:
In the average HTML document, you have a sea of similar elements—even a complex page often boils down to just headings and paragraph elements. This page has a general introduction followed by a series of book reviews. The page's author marked up the general introduction, the author credits, and the book summaries with `<p>` elements, but these components shouldn't all have the same formatting because they represent different types of content. A better approach is to format the different types of content (title, author, and description) in different ways.

First, it makes sense to lock down the standard details. For example, you might decide to set a single site-wide font, some basic paragraph spacing, and a default style for level-1 headings. Here's the part of the style sheet that makes that happen:

```
/* Set the font for the whole page. */
body {
    font-family: Georgia, serif;
}

/* Set some standard margins for paragraphs. */
p {
    margin-top: 2px;
    margin-bottom: 6px;
}

/* Format the heading with a background color. */
h1 {
    background-color: #FDF5E6;
    padding: 20px;
    text-align: center;
}
```

This style sheet includes three type selector rules. The first formats the `<body>` element, thereby applying the same font to the whole page. The second gives every `<p>` element the same margins, and the third changes the alignment and background color of `<h1>` headings. All these rules use type selectors, and are relatively straightforward.

Note: This example introduces another feature—CSS comments. CSS comments don't look like HTML comments. They always start with the characters `/*` and end with the characters `*/`. Comments let you document what each class selector represents. Without comments, it's all too easy to forget what each style rule does in a complicated style sheet, particularly when you use class selectors.

With these basic rules out of the way, it's time to move on to the real work of style sheet creation—writing the class rules. As you learned on page 147, class rules give you the flexibility to apply style sheet formatting to any element, and to as many elements as you want. They also help you think in a more logical, structured way about your web pages. Instead of focusing on the HTML tags you're using, classes help you focus on the different types of information you present.

For example, the review page breaks down to several ingredients: a review heading, a review byline (stating the author of the review), and the actual review text. Once you mentally divide your page into these sections, you can add class rules that format them.

For example, you might use a rule like this to format the review heading:

```
/* Make the review headings blue. */  
h2.review {  
    font-size: 100%;  
    color: blue;  
    margin-bottom: 0px;  
}
```

This rule applies to `<h2>` elements only, and only `<h2>` elements with the class name `review`. This technique makes sense because the website is sure to include other level-2 headings that don't correspond to review titles, and so need different formatting.

Tip: Remember, the point of the class name is to provide a succinct description of the type of content you want to format. In this example, the class name is `review`, because you're going to apply this style to the headings of book reviews. Good class names describe the *function* of the class rather than its appearance. For example, `WarningNote` is a good class name, while `BoldRedArialBox` isn't. The problem with the latter is that it won't make sense if you decide to change the formatting of your warning note box (for example, giving it red lettering).

Lastly, the style sheet defines two classes that format paragraph elements—one for the author byline, and one for the review text:

```
/* Make the bylines small and italicized. */  
p.byline {  
    font-size: 65%;  
    font-style: italic;  
    border-bottom-style: outset;  
    border-bottom-width: 1px;  
    margin-bottom: 5px;  
    margin-top: 0px;  
}  
  
/* Make book reviews a little smaller and justified. */  
p.review {  
    font-size: 83%;  
    text-align: justify;  
}
```

Now you simply need to apply your carefully structured style sheet to the review page. Here's the HTML markup that uses its classes. (To save space, most of the text is left out, but the essential structure is there.)

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
    <title>The Pessimist</title>  
    <link rel="stylesheet" href="PessimistReviews1.css" />  
</head>
```

```
<body>
  <h1>The Pessimist's Review Site</h1>
  <p>...</p>
  <p>...</p>
  <h2 class="review">How To Lose Friends and Fail in Life</h2>
  <p class="byline">Chris Chu</p>
  <p class="review">...</p>
  <h2 class="review">Europe 2009: Great Places to Miss</h2>
  <p class="byline">Antonio Cervantes</p>
</body>

</html>
```

Figure 6-19 shows the result.

The screenshot shows a Mozilla Firefox window displaying a local file at `file:///C:/Creating a Website/Chapter 6/PessimistReviews_Sty`. The title bar says "The Pessimist - Mozilla Firefox". The main content area displays the following:

The Pessimist's Review Site

Here you'll learn about the greatest unpublished books ever (not) written. Be sure to support this fictional Web site with contributions to the author. PayPal is accepted, but unmarked paper envelopes are preferred.

The reviews on this Web site do not correspond to reality. Any correspondence, real, imagined, or conjured up after a week of sleepless nights and heavy caffeine abuse is purely coincidental.

How To Lose Friends and Fail in Life
Chris Chu

Tired of sabotaging yourself endlessly? With this book, the author Chris Chu explains how to level the playing field and take on a challenge you can really master. So throw away those old self-help books and start accomplishing something!

Europe 2011: Great Places to Miss
Antonio Cervantes

Europe is brimming with world class attractions: glorious art galleries, charming bed-and-breakfast inns, old school restaurants and much more. But who can afford it? This book carefully documents some of the best attractions across Europe, and provides detailed plans that explain how to miss them and keep your last few paltry cents in your pocket. Guaranteed to keep you out of debtor's prison when you return home. (You did buy a return ticket in advance, right?)

Done

Figure 6-19:
Class rules let you format different parts of a document differently, even if they use the same element (like the ever-common `<p>` element).

Saving Work with the <div> Element

It can get tedious applying the class attribute to every element you want to format in your web page. For example, for every book review that has four or five paragraphs after the byline, you need to add the `class="review"` attribute to each paragraph. Fortunately, you can use a great shortcut, courtesy of the `<div>` element.

You may remember the `<div>` element from the previous chapter (page 117). It's a block element that lets you group together arbitrary sections of your web page. You can group as many elements with the `<div>` element as you want, including headings, paragraphs, lists, and more.

Thanks to style sheet inheritance (page 145), if you apply a class name to a `<div>` element, inheritance automatically applies the defined style to all the nested elements. So imagine you change this style sheet rule:

```
p.review {...}
```

to this:

```
div.review {...}
```

You can take this four-paragraph review:

```
<h2 class="review">...</h2>
<p class="byline">...</p>
<p class="review">...</p>
<p class="review">...</p>
<p class="review">...</p>
<p class="review">...</p>
```

and simplify your markup like this:

```
<div class="review">
  <h2 class="review">...</h2>
  <p class="byline">...</p>
  <p>...</p>
  <p>...</p>
  <p>...</p>
  <p>...</p>
</div>
```

You still need the class attribute for the heading and the byline, which gets its own distinct formatting. But the other paragraphs automatically inherit their formatting from the `<div>` that wraps them. And although there are some style properties (like `margin` and `padding`) that don't support inheritance, most do.

Figure 6-20 shows the result.



Figure 6-20:
In this example, the HTML markup wraps each review in a `<div>` element, which applies a background color and some borders, separating the reviews from the rest of the page. Techniques like these can help organize dense pages with lots of information.

The `<div>` element is a great way to save loads of time. Web experts use it regularly. But you can still improve your markup with one contextual selector, as described next.

Saving Work with Contextual Selectors

A *contextual selector* is stricter than an ordinary type selector. Whereas a type selector matches an element, a contextual selector matches an element *inside another element*. To understand the difference, take a look at this type selector:

```
b {  
    color: red;  
}
```

This selector formats all bold text in red. But what if you want to work on only bold text that appears inside a bulleted list? You can do that using the following contextual type selector, which finds unordered list elements (``) and then hunts for bold elements inside of them. If it finds any, it makes the bold text red:

```
ul b {  
    color: red;  
}
```

To create a contextual type selector, you simply put a space between the two elements.

Contextual selectors are useful, but thinking through the different possibilities for combining elements can get a little dizzying. You'll see the real benefit of a contextual selector when you use one to match a specific type of element inside a specific type of *class*. For example, consider what happens if you take this style sheet rule:

```
h2.review {...}
```

and change the selector to this:

```
div.review h2 {...}
```

The first part of this selector finds all the `<div>` elements in your page. The second part limits those matches to `<div>` elements with the class name *review*—which is exactly one. The third and final part of the selector locates the `<h2>` elements inside the `<div>`. The end result is that every level-2 review heading gets the appropriate formatting, while headings in the rest of the page are left alone.

Best of all, you remove the class attribute from the `<h2>` element, leaving the following, simpler markup:

```
<div class="review">  
    <h2>...</h2>  
    <p class="byline">...</p>  
    <p>...</p>  
    <p>...</p>  
    <p>...</p>  
    <p>...</p>  
</div>
```

This technique is a wildly popular way to define different formatting rules for different sections of a page. It also makes a particularly nice fit with the CSS-based layout techniques you'll learn about in Chapter 9, which use `<div>` elements to create and position separate panels of content.

POWER USER'S CLINIC

Creating a Style Sheet for Your Entire Website

Class rules aren't just useful for separating different types of content. They're also handy if you want to define rules for your entire site in a single style sheet.

In a typical website, you have pages or groups of pages you want to format differently. For example, you might have a page with your résumé, several pages chronicling your trip to Guadeloupe, and another group of pages that make up an online photo gallery. Rather than create three style sheets, you can create a single style sheet that handles everything. The trick is to use different class names for each section. In other words, you'll create a résumé class, a trip diary class, and a photo gallery class. Here's a basic outline of this approach:

```
/* Used for the resume pages. */  
.resume { ... }  
.resume h1 { ... }
```

```
.resume h2 { ... }
```

```
/* Used for the trip diary pages. */  
.trip { ... }  
.trip h1 { ... }  
.trip h2 { ... }
```

```
/* Used for the online photo gallery. */  
.gallery { ... }  
.gallery h1 { ... }  
.gallery h2 { ... }
```

Obviously, each page will use only a few of these rules. However, it's often easier to maintain your site when you keep your styles together in one place.

Adding Graphics

It's safe to say that the creators of the Internet never imagined it would look the way it does today—thick with pictures, ads, videos, and animated graphics. They designed a meeting place for leading academic minds; we ended up with something closer to a Sri Lankan bazaar. But no one's complaining, because the Web would be an awfully drab place without graphics.

In this chapter, you'll master the art of web graphics. You'll learn how to add ordinary images to a web page and to position them perfectly. You'll also consider what it takes to prepare pictures for the Web—or to find good alternatives online.

Understanding Images

To understand how images work on the Web, you need to know two things:

- They don't reside in your HTML files. Instead, you store each image as a separate file.
- To display pictures on a page, you use the `` element in your HTML document.

You'll use images throughout your site. You might even use them instead of ordinary text, if you need a special font.

Tip: If you're not sure whether a piece of content on a page is a graphic, try right-clicking it. If it's an image, most browsers give you a Save Picture As option in a pop-up menu.

The Element

Pictures appear on your web pages courtesy of the `` element, which tells a browser where to find them. For example, here's an `` element that displays the file named `photo01.jpg`:

```

```

Pictures are standalone elements (page 107), which means you don't need to include separate start and end tags in the element. Instead, you include the slash (/) character at the end of the tag, just before the closing angle bracket.

Pictures are also inline elements (page 107), which means you put them *inside* other block elements, like paragraphs:

```
<p></p>
```

When a browser reads this `` element, it sends out a request for the `photo01.jpg` file. After retrieving it, the browser inserts the file into the page where you put the `` element. If the image file is large or the Internet connection is slow, you might actually see this two-stage process take place, because smaller page components, like text, will appear before the image does.

Tip: You usually want to organize your site's many files by putting images in a subfolder of the folder that holds your web pages. You'll learn how to do this in Chapter 8.

Although it may seem surprising, the `` element is the only piece of HTML you need to display a picture. But to get the results you want, you must understand a few more issues, including how to use alternate text, modify the size of your images, choose a file format, and align your images with other content on a page.

Alternate Text

A browser can display an `` element as long as it has a `src` attribute. However, there's one other attribute that's strongly encouraged. It's the `alt` attribute, which represents the alternate text a browser displays if it can't display the image itself.

Here's an example:

```

```

Alternate text proves useful not only in the above circumstance, but in several other cases as well, including when:

- A browser doesn't support images (this is understandably rare these days, but the text-only Lynx browser is still kicking around on some old Unix systems).
- A web visitor switches off his browser's ability to display pictures to save page-download time (this isn't terribly common today, either).

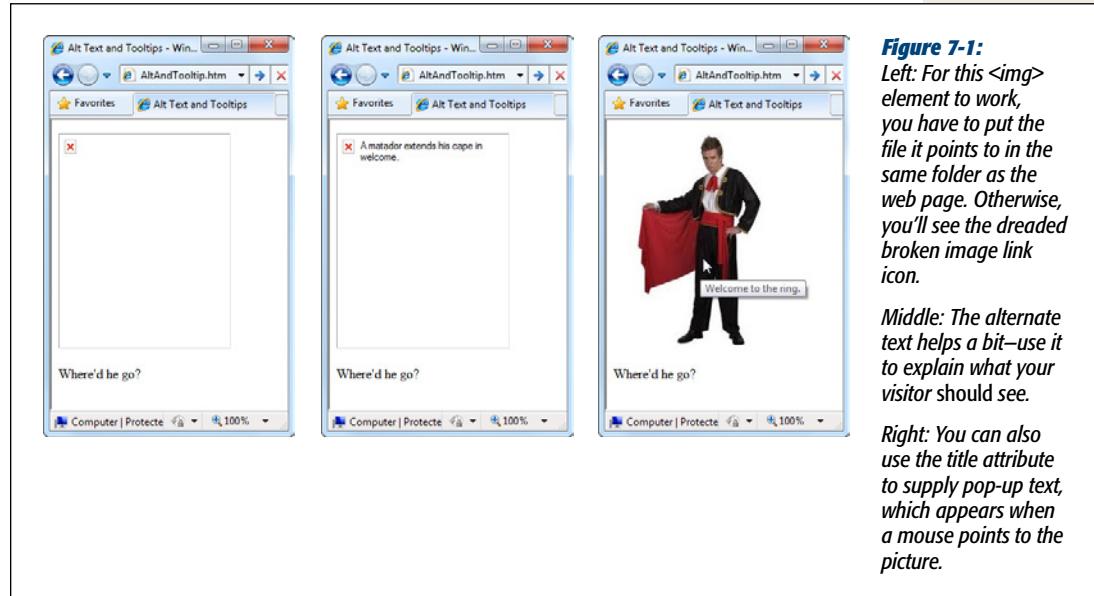
- A browser requests a picture, but can't find it. (Perhaps you forgot to copy it to your web server?)
- The visitor is viewing-impaired and uses a *screen-reading* program (a program that "speaks" text, including the words in an alt attribute).
- A search engine (like Google) analyzes a page and all its content so it can index the content in a search catalog.

The last two reasons are the most important. Web experts always use meaningful text when they write alt descriptions to ensure that screen readers and search engines interpret the corresponding pictures correctly.

Don't confuse alternate text with pop-up text, which is an optional message that appears when a website visitor moves her mouse cursor over an image (see Figure 7-1). To add pop-up text, use title attribute:

```

```



Different browsers display alt text differently. Internet Explorer shows alt text in an image box with a missing picture icon (see Figure 7-1, middle). But Firefox displays the alt text as a paragraph of ordinary content in the page, with no picture box or missing picture icon.

Picture Size

When you start thinking about the size of your images, remember that the word *size* has two possible meanings: it can refer to the dimensions of the picture (how much screen space it takes up on a web page), or it can signify the picture's file size (the number of bytes required to store it). To web page creators, both measures are important.

Picture dimensions are noteworthy because they determine how much screen real estate an image occupies. Web weavers measure graphics in units called pixels. A pixel represents one tiny dot on a computer screen. The web world doesn't work with fixed units like inches and centimeters, because you never know how large your visitor's monitor is, and therefore how many pixels it can cram in. (Chapter 9 has a detailed discussion of screen size and how to design your pages to satisfy the largest number of potential viewers.)

File size is also important because it determines how long it takes to send a picture over the Internet to a browser. Large pictures can slow down a web page, especially when it includes lots of pictures and you're sending it to someone who has an unusually slow Internet connection. (To understand file size and how you control it, you need to understand the different image file formats web browsers use, a topic discussed in the next section.)

Interestingly, the `` element lets you resize a picture through its optional `height` and `width` attributes. Consider this element:

```

```

In this line of code, you give the picture a width of 100 pixels and a height of 150 pixels. If this doesn't match the real dimensions of your source picture, browsers stretch and otherwise mangle the image until it fits the size you set (see Figure 7-2).

Note: Approach height and width attributes with caution. Sometimes, novice web authors use them to make *thumbnails*, small versions of large pictures. But using the height and width attributes to scale down a large picture comes with a performance penalty—namely, the browser still needs to download the original, larger image, even though it displays it at a smaller size. On the other hand, if you create thumbnails in a graphics editor like Photoshop, you can save them with smaller file sizes, ensuring that your pages download much speedier.

Many web page designers leave out image height and width attributes. However, experienced web developers sometimes add them using the *same* dimensions as the actual picture. As odd as this sounds, there are a couple of good reasons to do so.

First, when you include image size attributes, browsers know how large a picture is and can start laying out a page even as the graphic downloads (see Figure 7-1, left). On the other hand, if you don't include the height and width attributes, the browser

won't know the dimensions of the picture until it's fully downloaded, at which point it has to rearrange the content. This is potentially distracting if your visitors have slow connections and they've already started reading the page.

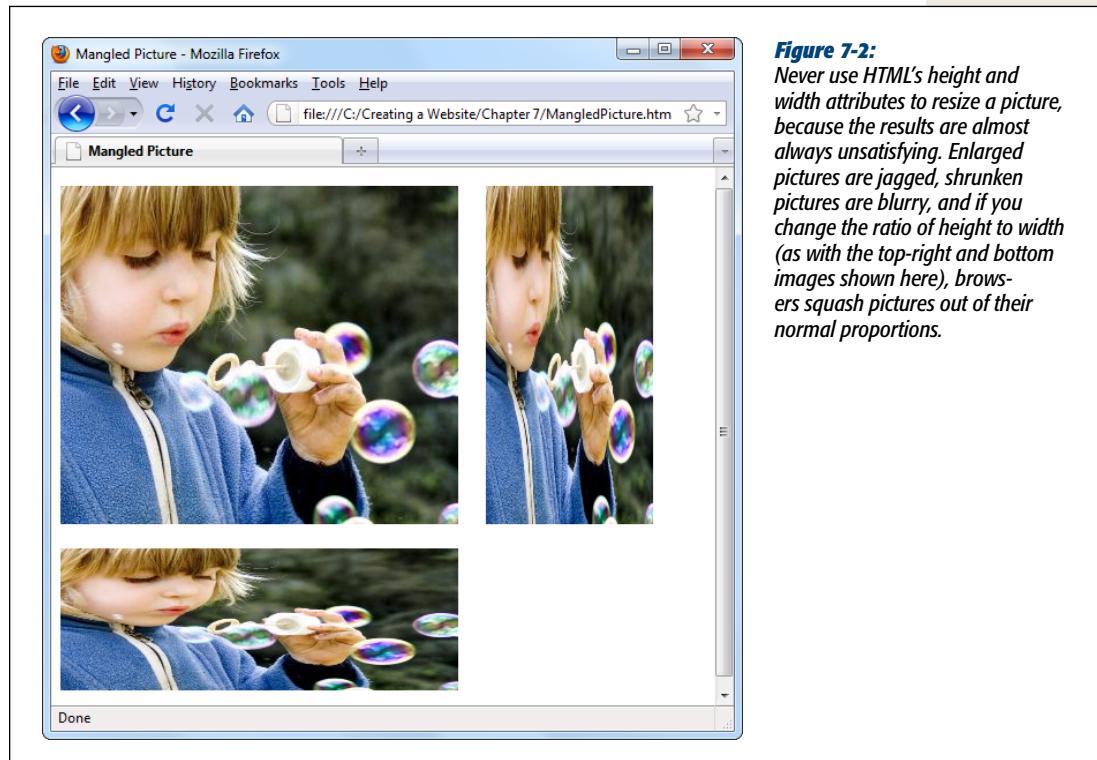


Figure 7-2:

Never use HTML's height and width attributes to resize a picture, because the results are almost always unsatisfying. Enlarged pictures are jagged, shrunken pictures are blurry, and if you change the ratio of height to width (as with the top-right and bottom images shown here), browsers squash pictures out of their normal proportions.

The second reason to use size attributes is because they control the size of the picture box if the browser can't download the image (see Figure 7-1, middle). However, you shouldn't rely on this trick, because it won't work in Firefox. That's because Firefox ignores the height and width attributes for missing pictures, and doesn't show an image box. (To really prevent a missing picture from scrambling your layout, you should carve your pages into separate sections using `<div>` elements, and position them with style sheets, as described in Chapter 9.)

So should you use the height and width attributes? It's up to you, but they're probably more trouble than they're worth for the average website. If you use them, you need to make sure to update them if you change the size of your picture, which quickly gets tedious. Or, simplify your life by making all your pictures the same size, and create a style sheet rule that sets the height and width properties.

Note: Some HTML editors, like Expression Web, automatically add the height and width attributes when you insert a picture.

File Formats for Graphics

Browsers can't display every type of image. In fact, they're limited to just a few image formats, including:

- **JPEG** (pronounced "jay-peg") format is suitable for photos that can tolerate some loss of quality. (As you'll learn in a moment, the JPEG format shrinks down, or compresses, an image's file size so that it downloads more quickly.) JPEG doesn't work as well if your picture contains text or line art.
- **GIF** (pronounced "jif" or "gif") format is suitable for graphics with a very small number of colors (like simple logos or clip art). It gives terrible results if you use it to display photos.
- **PNG** (pronounced "ping") format is suitable for all kinds of images, but it doesn't always compress as well as JPEG. PNG is particularly good for small, sharp graphics (like logos) and today it's used as a more powerful replacement for the GIF standard.
- **SVG** (Scalable Vector Graphics) is an up-and-coming standard for *vector drawings* (for example, logos and figures that consist of text and shapes, rather than photographs). For the right type of art, SVG has a number of advantages, including its small size and flexibility. You can resize SVGs without losing detail or getting blurry images. However, Internet Explorer versions 8 and earlier don't support SVG, which means it's not yet a viable choice for web graphics. For that reason, this chapter doesn't discuss it.

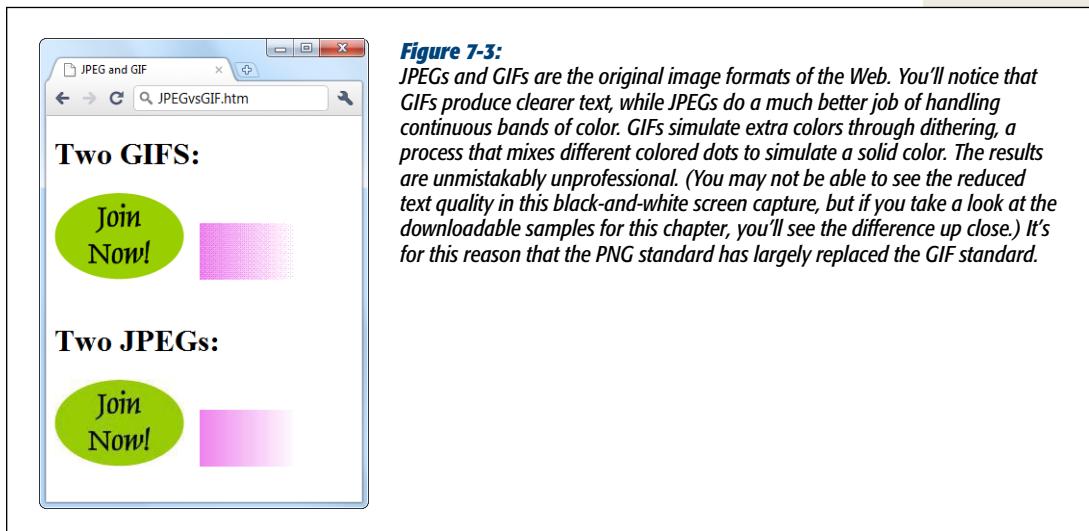
Note: You can learn more about SVG (and some of the workarounds that make it work, sort of, in Internet Explorer) at <http://tinyurl.com/2gy2vyg>.

Web page graphics use two types of compression: *lossy*, which compresses files to a greater degree than its alternative but also reduces image quality; and *lossless*, which preserves image quality but doesn't compress as much. For full details, see the box on page 191. Table 7-1 gives you a quick overview of the different image formats, and Figure 7-3 compares them in a web page.

Table 7-1. Image file formats for the Web.

Format	Type of Compression	Maximum Colors	Best Suited For
JPEG	Lossy	24-bit (16.7 million colors)	Photos.
GIF	Lossless	8-bit color (256 colors)	Simple logos, graphical text, and diagrams with line art.
PNG-8	Lossless	8-bit color (256 colors)	Not commonly used since it's similar to GIF (but it does have better transparency support).

Format	Type of Compression	Maximum Colors	Best Suited For
PNG-24	Lossless	24-bit (16.7 million colors)	Images that would normally be GIF files, but need more colors.
SVG	Lossless (but it's optional, as SVG data is already quite small)	24-bit (16.7 million colors)	Art drawn in an illustration program. However, versions of IE before IE 9 don't support it.

**Figure 7-3:**

JPEGs and GIFs are the original image formats of the Web. You'll notice that GIFs produce clearer text, while JPEGs do a much better job of handling continuous bands of color. GIFs simulate extra colors through dithering, a process that mixes different colored dots to simulate a solid color. The results are unmistakably unprofessional. (You may not be able to see the reduced text quality in this black-and-white screen capture, but if you take a look at the downloadable samples for this chapter, you'll see the difference up close.) It's for this reason that the PNG standard has largely replaced the GIF standard.

Choosing the right image format

It's important to learn which format to use for a given task. To help you decide, walk through the following series of questions.

Is your picture a hefty photo or does it have fine gradations of color?

YES: JPEG is the best choice for cutting large, finely detailed pictures down to size. Depending on the graphics program you use, you may be able to choose how much compression you want to apply.

Does your picture have sharp edges and need more than 256 colors?

YES: PNG is the best answer here. It supports full color, gives you lossless compression, and you don't lose any detail. If your picture has a limited number of colors (256 or less), you can use GIF instead of PNG, but there's no reason to prefer one over the other.

Note: Some browsers give you a few more options, but you're better off steering away from them to ensure widest browser compatibility. For example, Internet Explorer supports bitmaps (image files that end with the .bmp file name extension). Don't ever use them—not only will they confuse other browsers, but they're also ridiculously large because the standard doesn't include compression.

Does your picture include a transparent area?

YES: Use PNG. As you'll learn on page 192, PNG has support for partial transparency, which allows you to blend transparent regions and create a more natural effect. GIF has a cruder all-or-nothing transparency feature, and JPEG doesn't offer any support for transparency.

Note: You read earlier that HTML keeps page elements like headlines in rectangular boxes. The same holds true for images—even a picture of a beach ball is enclosed in a rectangle. That's why transparency in images is so important. It lets you place an image on a page with that page's background showing through; the result is a page with a seamlessly integrated image.

FREQUENTLY ASKED QUESTION

Typical File Sizes for Images

How much disk space does a typical picture occupy?

There's no single answer because it depends on several factors, including the dimensions of the picture, the file format you use, the amount of compression you apply, and how well the picture responds to compression techniques. However, here are a few basic things to keep in mind.

The file size of a typical website logo is vanishingly small. For example, Google's signature logo clocks in nearly a mere 20 KB (it's a PNG file).

Photos can take up much more disk space. On the small side of the equation, a picture in an article on the *New York Times* website rarely uses more than 70 KB. On the larger side of things, a typical eBay may include a product picture that's 300 KB. At this size, the picture usually takes up a larger portion of your browser window. However, that's nothing compared to the size the picture would be if you weren't using compression. For example, even an ancient 1-megapixel camera can take a raw, uncompressed picture of about 3,000 KB. In a web page, you can compress this to 300 KB or less by using the JPEG file format with a lower quality level.

Compression

Once upon a time, web connections were slow and web designers spent hours agonizing over the size of every image on their websites. Today, image size is a minor consideration, and pages are often clogged with animated advertisements and live video. However, professional websites still take the time to slim down their graphics. Doing so ensures that the website works well for everyone—even, say, visitors in a country where broadband connections are hard to find, or Westerners suffering with a sketchy wireless connection at the local coffee shop. Keeping things small also suits mobile devices, whose owners often need to pay based on the amount of data they download each month.

To keep your website as fast, lightweight, and efficient as possible, follow this advice:

- Keep your pictures small. If you really must fill the browser window, create a smaller version of the picture, put that on the page, and make it a link (page 213). Then, when someone clicks the picture, you can show a new page with the full-sized image.

- Use the right image format. For large photos, that's JPEG.
- To get better compression, lower the quality level of your JPEGs. But try it out first, to make sure you can tolerate the loss in detail. As you compress a JPEG image, you introduce various problems collectively known as *compression artifacts*. The most common artifacts are blocky regions of an image, halos around edges, and a general blurriness. Some pictures exhibit these flaws more than others, depending on the image's amount of detail.

Note: Most graphics programs let you choose how much you compress a picture, and many even let you preview the result before you save anything.

UP TO SPEED

How Compression Works

All three of the common web image formats use *compression* to shrink picture information. However, the type of compression you get with each format differs significantly.

The GIF and PNG formats support *lossless compression*, which means there's no *loss* of any information from your picture. Lossless compression uses a variety of techniques to perform its space-shrinking magic—for example, it might find a repeating pattern in the file, and replace each occurrence of it with a short abbreviation. When the browser decompresses your file, it gets all the original image data back.

The JPEG format uses *lossy compression*, which means it discards (or *loses*) some information about your picture.

As a result, your picture's quality diminishes, and there's no way to get it back to its original tip-top shape. However, the JPEG format is crafty, and it tries to trick your eye by discarding information that doesn't harm the picture that much. For example, it might convert slightly different colors to the same color, or replace fine details with smoothed-out blobs, because the human eye isn't that sensitive to small changes in color and shape. Usually, the overall result is a picture that looks softer and (depending how much compression you use) more blurry. On the other hand, the size-shrinking results you get with lossy compression are more dramatic than those offered by lossless compression.

Putting Pictures on Colored Backgrounds

Graphics-editing programs always store image files as rectangles, even when the image itself isn't rectangular. For example, if you create a smiley face graphic, your editing program saves that round illustration on a white, rectangular background.

If your page background is white as well, this doesn't pose a problem because the image's background blends in with the rest of your page. But if your page has a different background color (page 149), you'll run into the graphical clunkiness shown in Figure 7-4.



Figure 7-4:

Left: When you place the picture on a page with a white background, the smiley face blends right in.

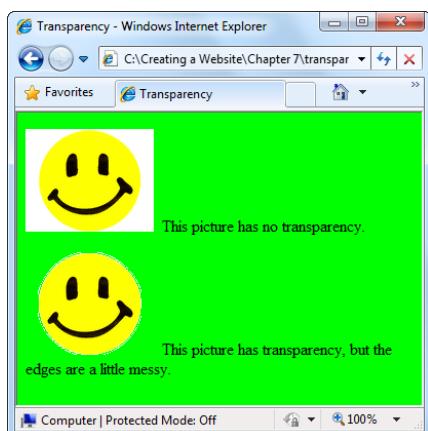
Right: With a non-white background, the white box around your picture is glaringly obvious.

Web designers came up with two solutions to this problem. One nifty idea is to use transparency, a feature common to both PNG and GIF graphics. The basic idea is that your image contains *transparent pixels*—pixels that don't have any color at all. When a browser comes across these, it doesn't paint anything. Instead, it lets the background of the page show through. To make part of an image see-through, you define a transparent color using your graphics program. In the example above, for instance, you'd set the white background of your smiley face graphic as the transparent color.

Although transparency seems like a handy way to make sure your image always has the correct background, in practice, it rarely looks good. The problem you usually see is a jagged edge where the colored pixels of your picture end and the web page background begins (see Figure 7-5).

The easiest way to fix this problem is to use the correct background color when you create your web graphic, instead of using transparency. In other words, when you draw your smiley-face image, give it the same background color as your web page. Your graphics program can then perform anti-aliasing, a technology that smoothes an image's jagged edges to make them look nice. That way, the image edges blend in well with the background, and when you display the image on your web page, it fits right in. The only limitation with this approach is its lack of flexibility. If you change your page color, you need to edit all your graphics.

Another approach is to use blended transparency (which is also known as *alpha blending*, as the alpha value is a number that tracks how transparent a pixel is). For example, instead of creating an image that cuts abruptly between colored pixels and transparent pixels, use one that blends the edge out using a fringe of semitransparent pixels. This feature sounds great, and is supported for PNG graphics. However, you'll need to have serious Photoshop skills to create this effect. (If you want to go this route, try starting with the barebones tutorial at <http://tinyurl.com/ypf78g>.) Sadly, this is the price of creating polished web graphics.

**Figure 7-5:**

The picture at the bottom of this page uses transparency, but the result—a jagged edge around the smiley face—is less than stellar. To smooth this edge, graphics programs use a sophisticated technique called anti-aliasing, which blends the picture color with the background color. Browsers can't perform this feat, so the edges they make aren't nearly as smooth.

UP TO SPEED

Graphics Programs

It's up to you to choose the format for your image files. Most good graphics programs (like Adobe Fireworks and Adobe Photoshop) save your documents in a specialized file format that lets you perform advanced editing procedures. Photoshop, for example, saves files in the .psd format. When you're ready to put your picture on a web page, you save a copy of the .psd file in a *different* format, one specially designed for the Web, like JPEG, GIF, or PNG. Usually, you do so by choosing File→Save As from the program's menu (although sometimes it's something a little different, like File→Export or File→Save For Web).

As a rule of thumb, you always need at least two versions of every picture you create: a copy in the original format your graphics program uses, and a copy in the JPEG, GIF, or PNG format you use on your website. You need to keep the original file so you can make changes whenever necessary and to make sure the image quality for future versions of the picture are as high as possible.

Once you choose the image format, your graphics program gives you a number of other options that let you customize details, like the compression level. At higher compression

levels, your image file is smaller but of lower quality. Some really simple image editors (like the Paint program that ships with Windows) don't let you tweak these settings, so you're stuck with the program's built-in settings.

Graphics programs usually come in two basic flavors—*image editors*, which let you retouch pictures and apply funky effects to graphics (like Adobe Photoshop), and *drawing programs*, which let you create your own illustrations by assembling shapes and text (like Adobe Illustrator). If you're editing pictures of the office party to cut out an embarrassing moment, an image editor makes sense. If you're creating a logo for your newly launched cookie company, you need a drawing program.

If you don't have the luxury of getting a professional graphics program, you can hunt for one on a shareware site like www.download.com. Two popular free image editors are GIMP (www.gimp.org), which supports all the major operating systems, and Paint.NET (www.getpaint.net), which is Windows only. You can also use the less-powerful but still quite impressive Phoenix and Vector online image editors at www.avaviary.com.

Images and Styles

The `` element supports a few optional attributes you can use to control an image's alignment and borders. But in the modern world, these attributes are obsolete, and you won't use them in this book. Instead, you'll learn the best way to position images—with style sheet rules.

The following sections describe your image-alignment options, and help you practice some of the style sheet smarts you picked up last chapter.

Inline Images in Text

If you don't take any extra steps, a browser inserts every image right into the flow of HTML text. It lines up the bottom of a graphic with the baseline of the text that surrounds it, as shown in Figure 7-6. (The baseline is the imaginary line on which a line of text sits.)



Figure 7-6:
Usually, you don't want a picture inside an ordinary line of text (unless it's a very small emoticon, like the ones you find in instant message programs). You can use paragraphs, line breaks, or tables to do a better job of separating images from your text.

You can change the vertical alignment of text using the `vertical-align` property. Specify a value of `top`, `middle`, or `bottom`, depending on whether you want to line the picture up with the top, middle, or bottom of the line of text.

Here's an example of an inline style that uses the `vertical-align` property to line a picture up with the top of the line of text.

```

```

This technique is worthwhile if you're trying to line up a very small picture, like a fancy bullet. But it doesn't work very well with large images. That's because no matter

which *vertical-align* option you choose, only one line of text can appear alongside the picture (as you can see in Figure 7-6). If you want to create floating pictures with wrapped text, see page 196.

Borders

In Chapter 6, you considered style properties that let you add and modify borders around boxes of text. It should come as no surprise that you can use these borders just as easily around images.

For example, here's a style that applies a thin, grooved border to all sides of an image:

```
img.BorderedImage {  
    border-style: groove;  
    border-width: 3px;  
}
```

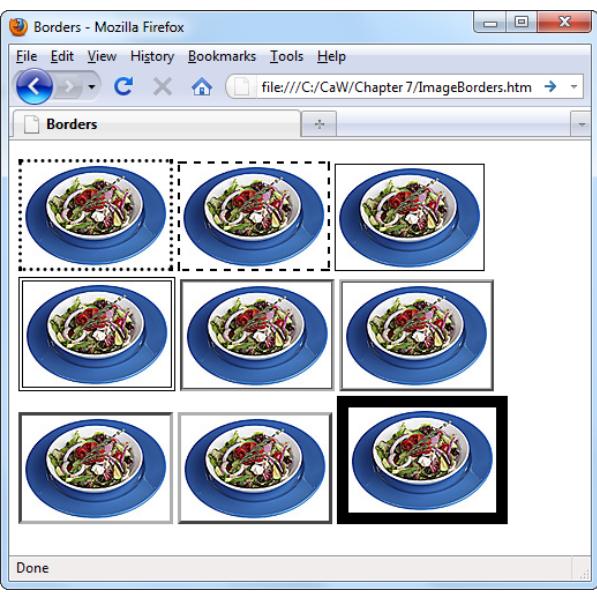
As with all style sheet rules, you need to place the rule in an internal style sheet in the current web page or in an external style sheet that your page uses (see page 136 for a discussion of the difference).

Notice that you give the style in this example a class name (*BorderedImage*). You don't want your browser to apply the style to every picture. Instead, you want to choose when to apply it using the *class* attribute:

```

```

Figure 7-7 shows the basic border styles. Remember, you can change the thickness of any border to get a very different look.



The screenshot shows a Mozilla Firefox window titled "Borders - Mozilla Firefox". The address bar displays "file:///C:/CaW/Chapter 7/ImageBorders.htm". The main content area contains a grid of nine images of a meal in bowls, each with a different border style applied via CSS. The borders include solid colors (blue, black), dashed lines, and various widths. A horizontal scroll bar is visible at the bottom of the browser window. To the right of the browser window, there is descriptive text about the figure.

Figure 7-7:
This example shows several inline images in a row, separated from one another with a single space. Each image in this example is the same, but sports a different border. The browser fits all the pictures it can on the same line. Upon reaching the right edge of the browser window, it wraps to the next line. If you resize the window, you'll see the arrangement of the pictures change.

Wrapping Text Around an Image

Using inline images is the simplest way to add pictures to your pages, but they have a downside: they pop up in the middle of text. To prevent this from happening, you can separate your pictures and text using paragraph elements (`<p>`), line breaks (`
`), horizontal rules (`<hr>`), and other divisions. You might decide, for example, to put a picture between two paragraphs of text, like this:

```
<p>This paragraph is before the picture.</p>
<p></p>
<p>This paragraph is after the picture.</p>
```

Inline images are locked into place. They never move anywhere you don't expect.

Sometimes, however, you want a different effect. Instead of separating images and text, you want to put them *alongside* each other. For example, you may want your text to wrap *around* one side of a picture.

Images that have text wrapped on one side or the other are called *floating* images, because they float next to an expanse of text (see Figure 7-8). You create floating images using a CSS property named `float`. You set the value of the `float` property to either left or right, which lines up the image on either the left or right edge of the text.

```
img.FloatLeft {
    float: left;
}
```

Notice that this example uses a class name. You probably don't want every image on your web page to float, so it's always a good idea to use a class name. Here's an `` using the above class, followed by some text:

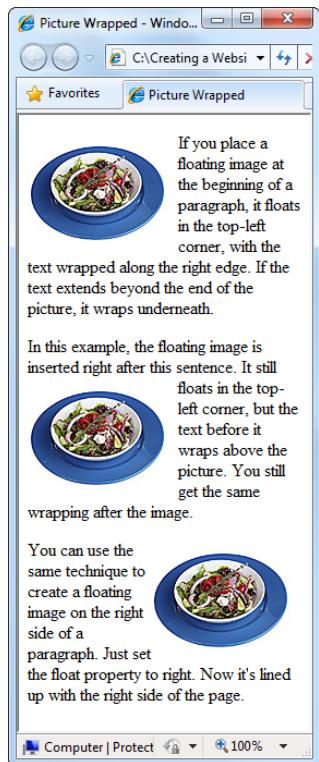
```
<p>
    
    If you place a floating image at the beginning of a paragraph,
    it floats in the top-left corner, with the text wrapped along
    the right edge.
</p>
```

When you set the `float` attribute, it makes sense to adjust the image's margin settings at the same time, so you have a little breathing room between your image and the surrounding text:

```
img.FloatLeft {
    float: left;
    margin: 10px;
}
```

Figure 7-8 shows several floating images.

Note: To get floating text to work the way you want, always put the `` element just *before* the text that should wrap around the image.

**Figure 7-8:**

Remember, all image files are really rectangles that include the surrounding white space (see Figure 7-4). As a result, the browser wraps text around the borders of these invisible squares.

Wrapping text can get a little tricky, because the results you get depend on the width of the browser window. For example, you might think your text is long enough to wrap around a graphic, but in a wide window, that might take up just a few short lines, letting the rest of the page's content bump into your floating graphic, which isn't what you want (see Figure 7-9).

To prevent this from happening, you can put your images in different containers, which is like having different cells in a table. Alternatively, you can manually stop your browser from wrapping text at any point using the *clear* property in a line break (`
`) element:

```
<br style="clear: both;" />
```

Place this line at the end of the wrapped paragraph, like so:

```
<p>
  
  Here is a paragraph with a floating image.
  <br style="clear: both;" />
</p>
<p>
```

```

This should be a separate paragraph with another
floating picture.
</p>
```

The *clear* property in a *
* element tells your browser to stop wrapping text, ensuring that the next paragraph starts after the floating picture (see Figure 7-9).

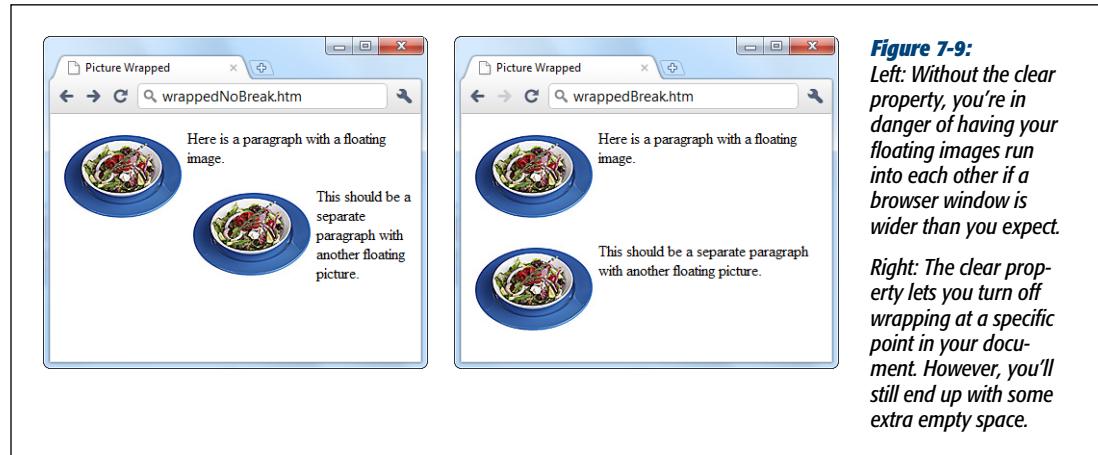


Figure 7-9:
Left: Without the *clear* property, you're in danger of having your floating images run into each other if a browser window is wider than you expect.

Right: The *clear* property lets you turn off wrapping at a specific point in your document. However, you'll still end up with some extra empty space.

Based on these examples, you might think that the *float* property sends a picture to the left or right side of a page, but that's not exactly what happens. Remember, in CSS, HTML treats each element on a page as a container. When you create a floating image, the image actually goes to the left or right side of its container. In the previous examples, this means that the image goes to the left or right side of a paragraph, because the containing element is a paragraph.

In the example above, the paragraph took up the full width of the page. But that doesn't have to be the case. You can use style rules to put a paragraph into a padded note box to get a completely different effect.

To try this out, you need to wrap the image and the paragraph in a *<div>* element, like this:

```
<div class="Box">
<p>
    
    <b>But Wait!</b> A tip box can interrupt the discussion
    to let you know just how good mixed veggies can taste.
    Of course, this tip box is really just an ordinary paragraph with
    the right border and margin style properties.
</p>
</div>
```

You can then apply a fancy border to the `<div>` element through a style rule:

```
div.Box {
    margin-top: 20px;
    margin-bottom: 10px;
    margin-left: 70px;
    margin-right: 70px;
    padding: 5px;
    border-style: dotted;
    border-width: 2px
}
```

Figure 7-10 shows the result.

The screenshot shows a Microsoft Internet Explorer window with the title "Picture Wrapped - Windows Internet Explorer". The address bar displays "C:\Creating a Website\Chapter 7\wrappedBox.htm". The page content is titled "Great Cooking Ideas" and includes a paragraph about exotic recipes. Below the paragraph are two floating image boxes. The first box contains a blue bowl of salad and the text "But Wait! A tip box can interrupt the discussion to let you know just how good mixed veggies can taste. Of course, this tip box is really just an ordinary paragraph with the right border and margin style properties." The second box also contains a blue bowl of salad and the text "Be careful, because if you have a short paragraph the picture might leak out of the box. It's an interesting effect, but hard to control." The browser interface shows standard controls like back, forward, and search at the top, and navigation, safety, and tools buttons below. At the bottom, it shows "Done", "Computer | Protected Mode: Off", and a zoom level of "100%".

Figure 7-10:

With crafty use of styles, you can lay out your pictures with the same flexibility you get when using styles to manipulate text.

Adding Captions

Another nice touch is to caption your pictures above or below an image. You can do this easily with inline images; just put a line of text immediately before or after the picture, separated by a line break. It's not so easy with a floating image, however. In this case, you need to have your image *and* the caption float in the same way.

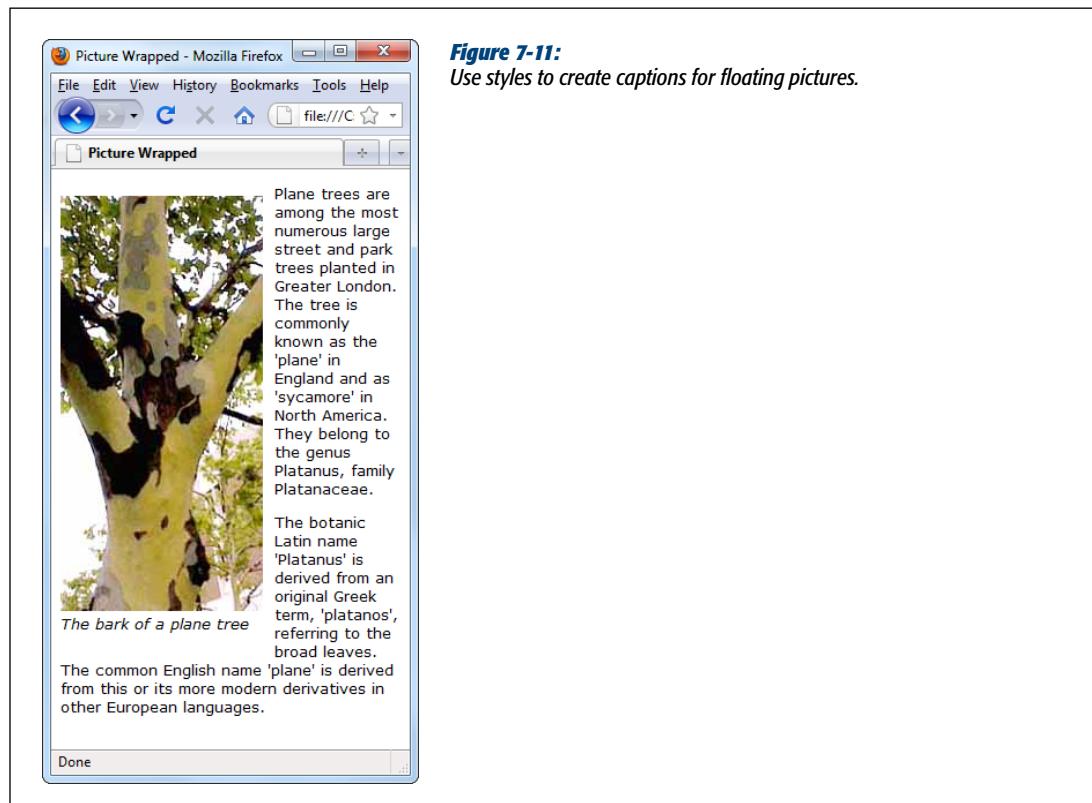
As it happens, the solution is quite easy. You simply take the `FloatLeft` style rule shown earlier and change the name from `img.FloatLeft` to `.FloatLeft` so you can use the rule with *any* element:

```
.FloatLeft {  
    float: left;  
    margin: 10px;  
}
```

Next, you wrap the `` element and your text into a `` element. You can then make the entire `` element float, by using the *FloatLeft* style rule:

```
<span class="FloatLeft">  
      
    <br />  
    <i>The bark of a plane tree</i>  
</span>
```

Figure 7-11 shows the result.



Note: You use a `` element in this example instead of a `<div>` element because you can put a `` element *inside* other block elements, like a paragraph. In other words, by using a `` element, you can easily put your floating picture-and-caption container inside one of your paragraphs.

HTML5 PREVIEW

A Better Way to Float a Figure

Figure 7-11 shows how you can use a floating box to hold a figure and some related text. This is a popular technique in the web design world, but it isn't perfect. One shortcoming is that it muddies up the structure of your web page. For example, it doesn't clearly indicate that the `` is meant to represent a figure, or that the text has anything to do with the image.

This might not strike you as a serious problem (and it isn't), but high-minded markup purists are busy planning a world where web pages have more structure, structure that helps programs better identify page elements—whether that program is a browser, search engine, accessibility tool, or something else altogether. (For more about this philosophical shift, refer back to page 105.)

HTML5 adds two new elements that let you define a clear, well-structured figure: `<figure>` and `<figcaption>`. The `<figcaption>` element wraps the text, and the `<figure>` element

wraps the whole shebang, picture and all. Here's how you would use the two to clean up the previous example:

```
<figure class="FloatLeft">
  
  <figcaption>The bark of a plane tree
  </figcaption>
</figure>
```

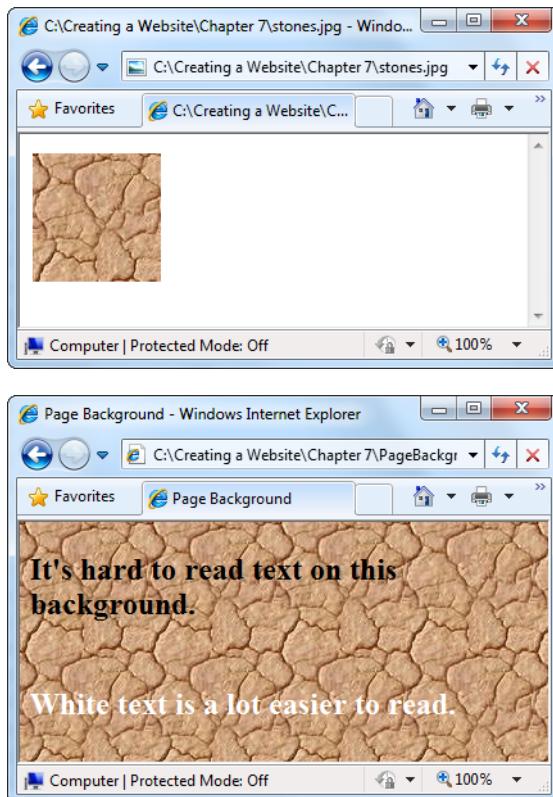
Neither new element applies any formatting, so you still need to use style sheet rules to get the appearance you want. But the real problem is that, like the other new HTML5 elements, you can't use CSS to style the `<figure>` and `<figurecaption>` elements in any version of Internet Explorer before IE 9. So unless you're willing to use a JavaScript hack (described on page 535), the Semantic-Web nerds will have to wait.

Background Images

CSS makes it possible to use an image as a page background, which is a particularly handy way to create “themed” websites. For example, you could use light parchment paper as a background for a literary site. A *Twilight* fan site might put a dark cemetery image to good use. Some people find the effect a little distracting, but it's worth considering if you want to add a really dramatic touch and you can restrain yourself from going overboard.

Tip: Background images can make your website seem tacky. Be wary of using them for a résumé page or a professional business site. On the other hand, if you want to go a little kitschy, have fun!

Web designers almost always choose to *tile* background images, which means the browser copies a small picture over and over again until the image fills the window (see Figure 7-12). You can't use a single image to fill a browser window because you have no way of knowing how wide and tall to make it, given people's variable browser settings. Even if you did have visitors' exact screen measurements, you'd need to create an image so ridiculously large that it would take an impractically long time to download.

**Figure 7-12:**

Top: Start with a small tile graphic with a stony pattern.

Bottom: Using style sheets, you can tile this graphic over the whole page. In a good tiled image, the edges line up to create the illusion of a seamless larger picture.

To create a tiled background, use the *background-image* style property. Your first step is to apply this property to the `<body>` element, so that you tile the whole page. Next, you need to provide the file name of the image using the form `url('filename')`, as shown here:

```
body {
    background-image: url('stones.jpg');
}
```

This tiles the image *stones.jpg* across a page to create your background.

Keep these points in mind when you create a tiled background:

- Make your background light, so the text displayed on top of it remains legible. (If you really have to go dark, you can use white, bold text so that it stands out. But don't do this unless you're creating a website for a trendy new band or you're opening a gothic clothing store.)
- Set the page's background color to match the image. For example, if you have a dark background picture with white text, make the background color black.

That way, if a browser can't download the background image, visitors can still see the text.

- Use small tiles to reduce the amount of time your visitors need to wait before they can see the page.
- If your tiled image has an irregular pattern, make sure the edges line up. The left edge should continue the right edge, and the top edge should continue the bottom edge. Otherwise, when the browser tiles your image, you'll see lines where it stitches the tiles together.

Tip: The Web is full of common background images, like stars, blue skies and clouds, fabric and stone textures, fires, dizzying geometric patterns, borders, and much more. You can find these by searching Google for "backgrounds," or head straight to the somewhat dated sites that specialize in downloadable backgrounds, like www.grsites.com/textures and www.backgroundarchive.com.

Background watermarks

Most websites tile a picture to create a background image, but that's not your only option. You can also take a single image and place it at a specific position on your page. Think, for example, of a spy site whose background image faintly reads "Top Secret and Confidential."

An inconspicuous single-image background like this is called a *watermark*. (The name stems from the process used to place a translucent logo on paper saturated with water.) To make a good watermark, use a background picture that's pale and unobtrusive.

To add a watermark to your page, use the same *background-image* property you learned about above. But you need to add a few more style properties to the mix (see Table 7-2). First, you have to use the *background-repeat* property to turn off tiling. At the same time, it makes sense to use the *background-position* property to align your picture to a side of the page or to its center.

Table 7-2. Background image properties.

Property	Description	Common Values	Can Be Inherited
<code>background-image</code>	The image file you want to use as your page background.	A URL pointing to the image file, as in <code>url('mypig.jpg')</code>	No ¹
<code>background-repeat</code>	Whether or not you tile the image to fill the page; you can turn off tiling altogether, or turn it off in one dimension (so that images tile vertically but not horizontally, for example).	<code>repeat, repeat-x, repeat-y, no-repeat</code>	No

Property	Description	Common Values	Can Be Inherited
background-position	Where you want to place the image. Use this only if you <i>aren't</i> tiling the image.	top left, top center, top right, center left, center, center right, bottom left, bottom center, bottom right	No
background-attachment	Whether you want to fix the image (or tiles) in place when a visitor scrolls the page.	scroll, fixed	No

¹ Background pictures aren't inherited (see page 145). However, if you don't explicitly assign a background color to an element, it's given a transparent background, which means the background of the containing element will show through.

Here's an example that places a picture in the center of a web page:

```
body {
    background-image: url('smiley.jpg');
    background-repeat: no-repeat;
    background-position: center;
}
```

Note: The center of your document isn't necessarily the center of your browser window. If you position your image in the center of a long web page, you won't see it until you scroll down.

You can also turn off an image's ability to scroll along with the rest of a page to get the rather odd effect of an image that's fixed in place (see Figure 7-13). For example, use this style to create a background image that sits squarely in the center of a window:

```
body {
    background-image: url('smiley.gif');
    background-repeat: no-repeat;
    background-position: center;
    background-attachment: fixed;
}
```

Techniques with Graphics

Now that you've mastered the `` element, it's time to learn a few tricks of the trade. In the following sections, you'll tour three common techniques that web developers everywhere use to create more polished pages.

Graphical Text

In Chapter 6 (page 165), you learned that using exotic fonts on web pages can be tricky. Not only do you need to think about font licensing, you also need to create several versions of each typeface to make sure your fancy font works in different browsers.

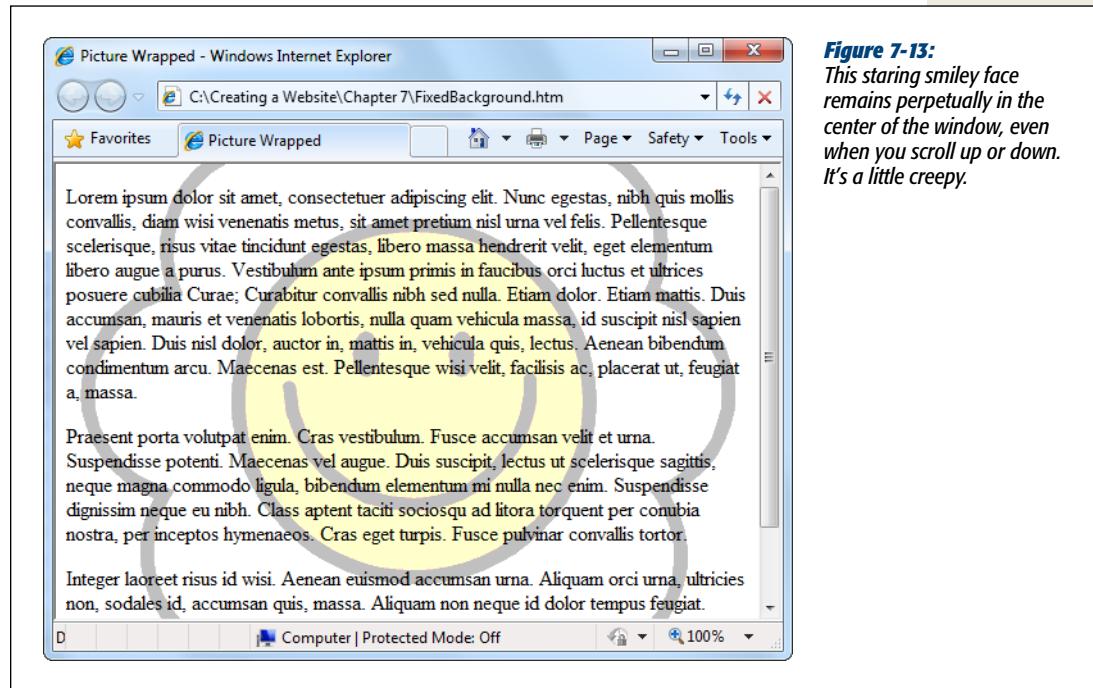


Figure 7-13:
This staring smiley face remains perpetually in the center of the window, even when you scroll up or down. It's a little creepy.

There's no way to get around this hassle when you have large blocks of text. However, enterprising web *artistes* sometimes put the short snippets of text they need for headings, buttons, and logos into picture files. That way, they get *complete* control of what the text looks like, and life is a lot less complicated.

Here's a high-level look at what you do to create small pictures with text:

1. Fire up your favorite image editor or drawing program.

Figure 7-14 shows an example with Adobe Illustrator.

2. Use a background color that matches your web page.

In some programs, the easiest way to fill a section with color is to draw a shape (like a rectangle), and then give it the proper "fill" color.

3. Choose your font, and then type the text over the background color.

4. Cut your image down to size.

Ideally, you want to make the image as small as possible without clipping off any text.

5. Save your picture.

GIF is a good format choice, but you'll need PNG if the image has more than 256 colors. Don't use JPEG, or your text will have blurred edges.

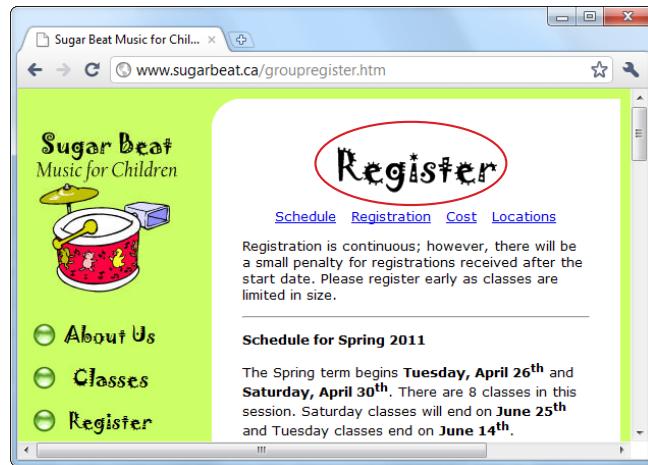
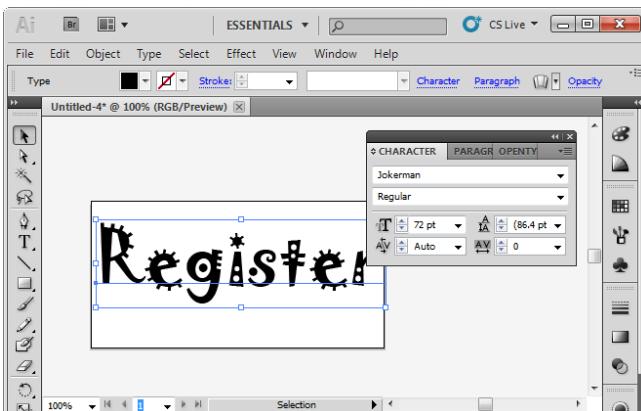


Figure 7-14:

Top: The final touches being made to a single-word heading in Adobe Illustrator.

Bottom: The final picture as it appears on a web page. The process of creating graphical text can be tedious, especially if you have a lot of headings to generate. But it's the most reliable way to bring funky fonts to the Web.

Graphical text is clearly unsuitable for large amounts of copy. For example, graphical text can't adjust itself to fit the width of a browser window or take into account your visitors' browser preference settings. There's also no way for a visitor to search your graphical text for specific words, for a screen reading program to interpret it, or for a web search engine to figure out what's on your site. If you decide to use graphical text, make sure you repeat the text with the *alt* attribute to lessen these issues.

Backgrounds for Other Elements

You don't need to apply a background to a whole page. Instead, you can bind a background to a single paragraph or, more usefully, to a `<div>` element, creating the same effect as a sidebar in a magazine. Usually, you want to add a border around this element to separate it from the rest of your web page. You might also need to change the color of the foreground text so it's legible (for example, white shows up better than black on dark backgrounds).

Here's an example of a background image you can use with any container element:

```
.pie {  
    background-image: url('pie.jpg');  
    margin-top: 20px;  
    margin-bottom: 10px;  
    margin-left: 70px;  
    margin-right: 70px;  
    padding: 10px;  
    border-style: double;  
    border-width: 3px;  
    color: white;  
    background-color: black;  
    font-size: large;  
    font-weight: bold;  
    font-family: Verdana, sans-serif;  
}
```

This style specifies a background image, sets the margins and borders, and chooses background and foreground colors to match.

Here's a `<div>` element that uses this style:

```
<div class="pie">  
    <p>Hungry for some pie?</p>  
</div>
```

Figure 7-15 shows the result.

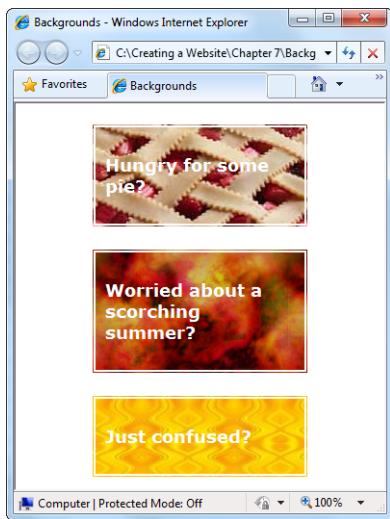
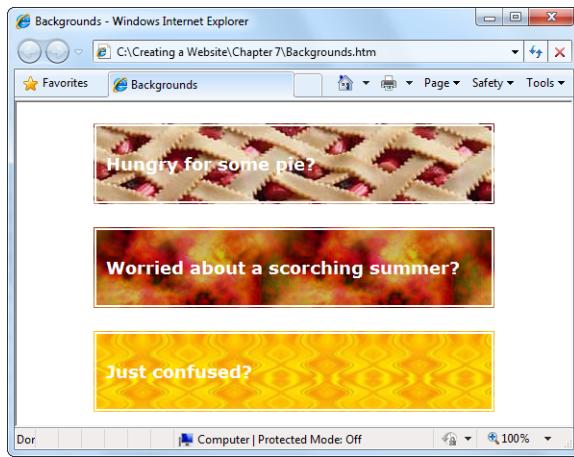


Figure 7-15:

Top: Using background images in small boxes is surprisingly slick.

Bottom: A particularly neat feature is the way the picture grows when you resize the page, thanks to tiling.



Graphical Bullets in a List

In Chapter 5, you learned how to use the `` element to create a bulleted list. However, you were limited to a small set of predefined bullet styles. If you look around the Web, you'll see more interesting examples of bulleted lists, including some that use tiny pictures as custom bullets.

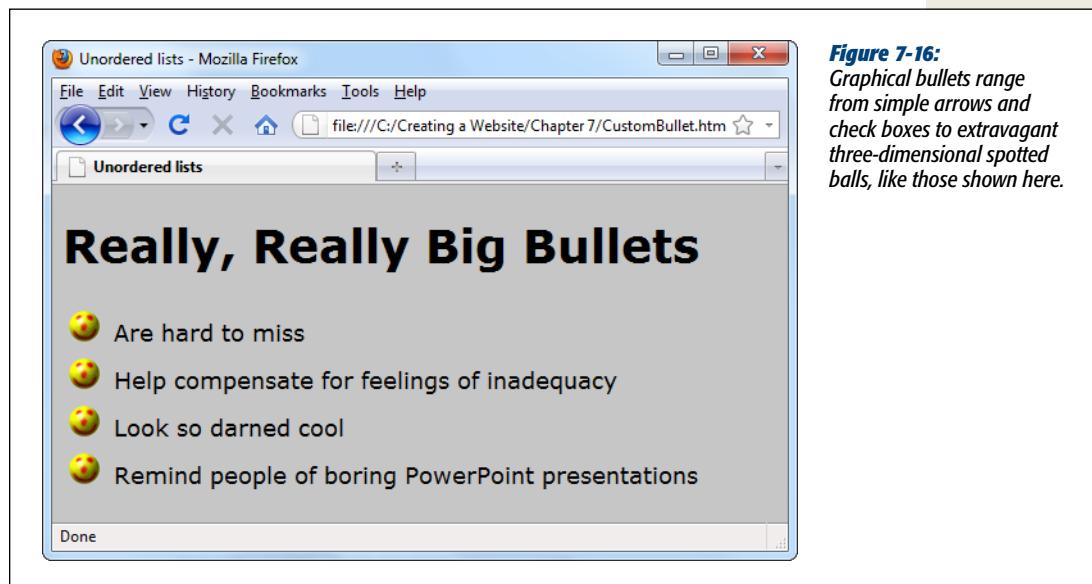
You can add custom bullets by hand using the `` element, but there's an easier option. You can use the `list-style-image` property to set a bullet image. Here's an example that uses a picture named `3Dball.gif`:

```
ul {
    list-style-image: url('3Dball.gif');
}
```

Once you create this style rule and put it in your style sheet, your browser automatically applies it to an ordinary bulleted list like this one:

```
<ul>
<li>Are hard to miss</li>
<li>Help compensate for feelings of inadequacy</li>
<li>Look so darned cool</li>
<li>Remind people of boring PowerPoint presentations</li>
</ul>
```

Figure 7-16 shows the result.



Finding Free Art

The Web is awash in graphics. In fact, finding a web page that isn't chock full of images is about as unusual as spotting Bill Gates in a dollar store. But how do you generate all the pictures you need for a graphically rich site? Do you really need to spend hours in a drawing program fine-tuning every picture you want? The answer depends on exactly what type of pictures you need, of course, but you'll be happy to hear that the Web is a great resource for ready-to-use pictures.

It's not hard to locate pictures on the Web. In fact, you can even use a handy Google tool to search for graphics on a specific subject (type <http://images.google.com> into your browser and search away). Unfortunately, *finding* an image usually isn't good enough. To use it without worrying about a nefarious lawyer tracking you down, you also need the *rights* to use the picture. If you get lucky, a website owner might grant

you permission to use a graphic after you send a quick email. But that's the exception rather than the rule.

Fortunately, photo enthusiasts have set up community sites where they post their pictures for the world to see—and on some of these sites, you can search for and reuse anything you want, for free. One of the best is Stock.XCHNG (pronounced “stock exchange,” after *stock photography*, the name for the vast catalogues of reusable pictures that graphic designers collect). To visit Stock.XCHNG, go to <http://sxc.hu>. Figure 7-17 shows a Stock.XCHNG search in progress.

Type in your search keywords

The screenshot shows a Mozilla Firefox browser window with the URL <http://www.sxc.hu/browse.php?ff=search&txt=paris+food&n=100&0By=0>. The search term 'paris food' is entered in the search bar. The results page shows 11 images. Red annotations provide the following information:

- A red box highlights the search bar with the text "Type in your search keywords".
- An arrow points to the sponsored images with the text "These are sponsored pictures that come with a price".
- An arrow points to the free images with the text "These are the free pictures".
- A red box highlights a specific image thumbnail with the text "Here's a thumbnail preview of the full picture".
- The text "This is the size of the full picture in pixels. You probably want to shrink it using a drawing program." is associated with the highlighted image.
- The text "Click here to see other pictures by this photographer" is located at the bottom left of the results page.

Figure 7-17:
Stock.XCHNG offers a searchable catalogue of well over 100,000 photos on every subject. Every day, eager photo enthusiasts upload their sometimes-striking work, including some of the images used in this book. In this figure, a search for “paris food” results in some interesting culinary treats.

Another good place for digging up free photos is Flickr (www.flickr.com). However, Flickr includes images that are both free to reuse and not. The best way to find ones that you can use is to perform an advanced search at www.flickr.com/search/advanced. Enter your search keywords (as you would in a normal search), but check the “Only search within Creative Commons-licensed content” setting (Figure 7-18). This makes sure that you’ll only find files you can snatch.

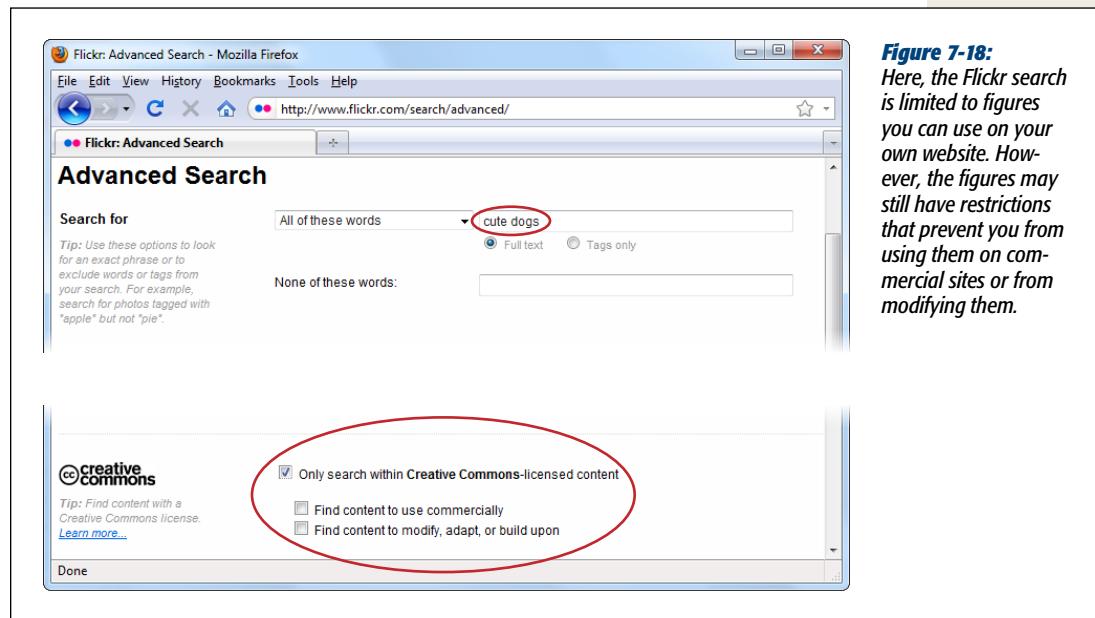


Figure 7-18: Here, the Flickr search is limited to figures you can use on your own website. However, the figures may still have restrictions that prevent you from using them on commercial sites or from modifying them.

Be aware, though, that content licensed with a Creative Commons license may apply one or two restrictions. For example, it may stipulate that you can't use the picture on a commercial website (at least not without getting permission from the picture owner). It may also restrict you from changing or editing the picture. When you find a picture you like, you can get the exact license details by scrolling down to the License box (which you'll find in the bottom-right corner of the page). Or, you can explicitly search for pictures that don't have these restrictions, using the additional settings shown in Figure 7-18.

If you can't find the picture you want at Stock.XCHNG or Flickr, you may never find it—at least not without going to a commercial site, like iStockPhoto (www.istockphoto.com), Fotolia (www.fotolia.com), or Dreamstime (www.dreamstime.com), all of which charge a few dollars for royalty-free images. But if you'd like to look at some other no-pay alternatives, check out the article on finding free photographs at <http://tinyurl.com/49yquv3>.

Tip: A lot of so-called “free” clip art sites are choked with ads and subscription demands. However, if you have a copy of Microsoft Office, you can download free clip art straight from the Office Online website. Head to <http://office.microsoft.com/images> to start searching. Depending on the image you pick, you may need to convert it to a web image format (like JPEG or PNG), but it's a small price to pay for access to a large, free clip art connection.

Linking Pages

So far in this book, you've worked on individual web pages. While creating a single page is a crucial first step in building a site, sooner or later you'll want to wire several pages together so a web trekker can easily jump from one page to another. After all, linking is what the Web's all about.

It's astoundingly easy to create links—officially called *hyperlinks*—between pages. In fact, all it takes is a single new element: the *anchor* element. Once you master this bit of HTML lingo, you're ready to start organizing your pages into separate folders and transforming your humble collection of standalone documents into a full-fledged site.

Understanding the Anchor

In HTML, you use the anchor element, `<a>`, to create a link. When a visitor clicks that link, the browser loads another page.

The anchor element is a straightforward container element. It looks like this:

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

You put the text a visitor clicks inside the anchor element:

```
<a>Click Me</a>
```

The problem with the above link is that it doesn't *point* anywhere. To turn it into a fully functioning link, you need to supply the URL of the destination page using an `href` attribute (which stands for *hypertext reference*). For example, if you want a link to take a reader to a page named *LinkedPage.htm*, you create this link:

```
<a href="LinkedPage.htm">Click Me</a>
```

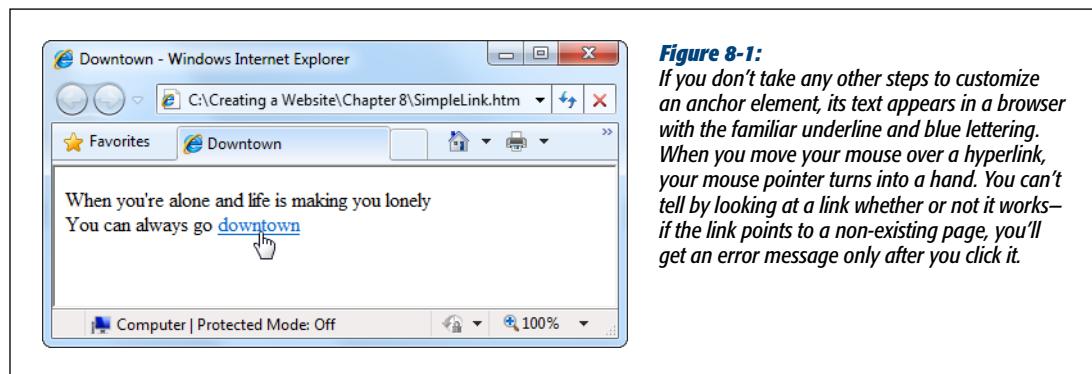
For this link to work, the *LinkedPage.htm* file has to reside in the same folder as the web page that contains the Click Me link. You'll learn how to better organize your site by sorting pages into different subfolders on page 217.

Tip: To create a link to a page in Expression Web, select the text you want to make clickable, and then press Ctrl+K. Browse to the correct page, and Expression Web creates the link. To pull off the same trick in Dreamweaver, select the text and press Ctrl+L.

The anchor tag is an inline element (page 107)—it fits inside any other block element. That means that it's completely acceptable to make a link out of just a few words in an otherwise ordinary paragraph, like this:

```
<p>  
    When you're alone and life is making you lonely<br />  
    You can always go <a href="Downtown.htm">downtown</a>  
</p>
```

Figure 8-1 shows this link in action.



However, it's worth noting that link text—that's the text inside the `<a>` element—is important. As you'll discover in Chapter 11, search engines pay extra attention to it. Getting it right increases the odds that your website will turn up as a search result and attract new visitors. Here are some examples of good, descriptive link text: “Products,” “Register,” “Our Policies,” and “Contact Me.” By comparison, unhelpful link text usually consists of one or two vague words in a sentence, like “click,” “here,” “more,” or “this link.”

Internal and External Links

Links can shuffle you from one page to another within the same website, or they can transport you to a completely different site on a far-off server. You use a specific type of link in each case:

- Internal links point to other pages on your site. They can also point to other types of resources on your site, as you'll see below.
- External links point to pages (or resources) on other websites.

For example, say you have two files on your site, a biography page and an address page. If you want visitors to go from your bio page (*MyBio.htm*) to your address page (*ContactMe.htm*), you create an internal link. Whether you store both files in the same folder or in different folders, they're part of the same website on the same web server, so an internal link's the way to go.

On the other hand, if you want visitors to go from your Favorite Books page (*Fav-Books.htm*) to a page on Amazon.com (www.amazon.com), you need an external link. Clicking an external link transports the reader out of your website and on to a new site, located elsewhere on the Web.

When you create an internal link, you should always use a *relative URL*, which tells browsers the location of the target page *relative to the current folder*. In other words, it gives your browser instructions on how to find the new folder by telling it to move down into or up from the current folder. (Moving *down into* a folder means moving from the current folder into a subfolder. Moving *up from* a folder is the reverse—you travel from a subfolder up into the parent folder, the one that *contains* the current subfolder.)

All the examples you've seen so far use relative URLs. For example, imagine you go to this page:

<http://www.GothicGardenCenter.com/Sales/Products.htm>

Say the text on the *Products.htm* page includes a sentence with this relative link to *Flowers.htm*:

Would you like to learn more about our purple
hydrangeas?

If you click this link, your browser automatically assumes that you stored *Flowers.htm* in the same location as *Products.htm*, and, behind the scenes, it fills in the rest of the URL. That means the browser actually requests this page:

<http://www.GothicGardenCenter.com/Sales/Flowers.htm>

HTML gives you another linking option, called an *absolute URL*, which defines a URL in its entirety, including the domain name, folder, and page. If you convert the URL above to an absolute URL, it looks like this:

Would you like to learn more about our purple hydrangeas?

So which approach should you use? Deciding is easy. There are exactly two rules to keep in mind:

- If you're creating an external link, you *must* use an absolute URL. In this situation, a relative URL just won't work. For example, imagine you want to link to the page *home.html* on Amazon's website. If you create a relative link, the browser

assumes that *home.html* refers to a file of that name on *your* website. Clicking the link won't take your visitors where you want them to go (and may not take them anywhere at all, if you don't have a file named *home.html* on your site).

- If you're creating an internal link, you really, really should use a relative URL. Technically, either type of link works for internal pages. But relative URLs have several advantages. First, they're shorter and make your HTML more readable and easier to maintain. More importantly, relative links are flexible. You can rearrange your website, put all your files into a different folder, or even change your site's domain name without breaking relative links.

One of the nicest parts about relative links is that you can test them on your own computer and they'll work exactly the same way as they would online. For example, imagine you developed the site *www.GothicGardenCenter.com* on your computer and you store it inside the folder *C:\MyWebsite* (that'd be *Macintosh HD/MyWebsite*, in Mac-ese). If you click the relative link that leads from the *Products.htm* page to the *Flowers.htm* page, the browser looks for the target page in the *C:\MyWebsite* (*Macintosh HD/MyWebsite*) folder.

Once you polish your work to perfection, you upload the site to your web server, which has the domain name *www.GothicGardenCenter.com*. Because you used relative links, you don't need to rewrite any of the links when you move your pages to the server. When you click a link, the browser requests the corresponding page in *www.GothicGardenCenter.com*. If you decide to buy a new, shorter domain name like *www.GGC.com* and move your website there, the links still work.

Note: Internet Explorer has a security quirk that appears when you test pages with external links. If you load a page from your hard drive, and then click a link that points somewhere out on the big bad Web, Internet Explorer opens a completely new window to display the target page. That's because the security rules that govern web pages on your hard drive are looser than those that restrict web pages on the Internet, so Internet Explorer doesn't dare let them near each other. This quirk disappears once you upload your pages to the Web.

FREQUENTLY ASKED QUESTION

Showing a New Window

How do I create a link that opens a page in a new browser window?

When visitors click external links, you might not want to let them get away from your site that easily. Web developers use a common trick that opens external pages in separate browser windows (or in a new tab, depending on the browser's settings). This way, your site remains open in the visitor's original window, ensuring she won't forget about you.

To make this work, you need to set another attribute in the anchor element—the *target*. Here's how:

```
<a href="LinkedPage.htm" target="_blank">  
Click Me</a>
```

The target attribute names the frame where a browser should display the destination page (you'll learn more

about frames in Chapter 10). The value *_blank* indicates that the link should load the page in a new, empty browser window.

Before you start adding the *target* attribute to all your anchors, it's important to realize that it may not always work. Some vigilant *pop-up blockers* intercept this type of link and prevent the new window from appearing altogether. (Pop-up blockers are standalone programs or browser features designed to prevent annoying pop-up ads from appearing.) Internet Explorer 6 (and later) includes its own pop-up blocker, but its standard settings allow links that use the *target=" _blank"* attribute.

Some people love the new-window feature, while others think it's an immensely annoying and disruptive act of website intervention. If you use it, apply it sparingly on the occasional link.

Relative Links and Folders

So far, all the relative link examples you've seen have assumed that both the *source page* (the one that contains the link) and the *target page* (the destination you arrive at when you click the link) are in the same folder. There's no reason to be quite this strict. In fact, your website will be a whole lot better organized if you store groups of related pages in *separate* folders.

Consider the website shown in Figure 8-2.

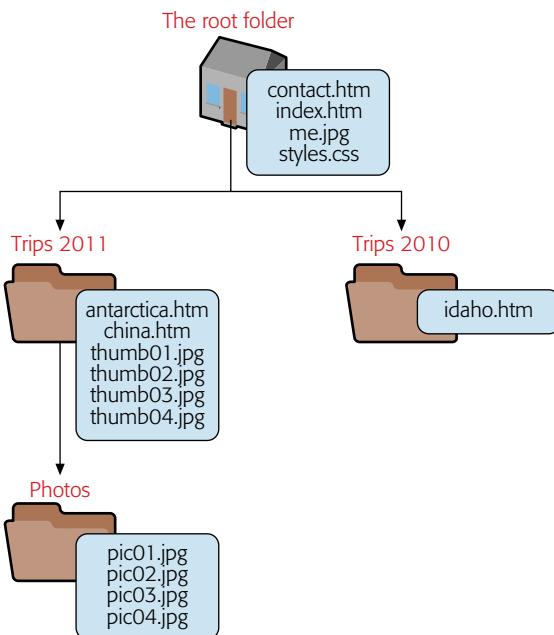


Figure 8-2:

This diagram maps out the structure of a very small website featuring photos taken on a trip. The root folder contains a style sheet used across the entire site (`styles.css`) and two HTML pages. Two subfolders, `Trips2010` and `Trips2011`, contain additional pages. The `Trips2011` folder holds thumbnail images of pictures taken on one of the trips. For each thumbnail, there's a corresponding full-size picture in the `Photos` subfolder.

Note: The root folder is the starting point of your website—it contains all your other site files and folders. Most sites include a page with the name `index.htm` or `index.html` in the root folder. This is known as the *default page*. If a browser sends a request to your website domain without supplying a file name, the server sends back the default page. For example, requesting `www.TripToRemember.com` automatically returns the default page `www.TripToRemember.com/index.htm`.

This site uses a variety of relative links. For example, imagine you need to create a link from the `index.htm` page to the `contact.htm` page. Both pages are in the same folder, so all you need is a relative link:

```
<a href="contact.htm">About Me</a>
```

You can also create more interesting links that move from one folder to another, which you'll learn how to do in the following sections.

Tip: If you'd like to try out this sample website, you'll find all the site files on the Missing CD page at www.missingmanuals.com/cds/caw3. Thanks to the magic of relative links, all the links will work no matter where on your computer you save the files, so long as you keep the same subfolders.

Moving down into a subfolder

Say you want to create a relative link that jumps from an *index.htm* page to a page called *antarctica.htm*, which you put in a folder named *Trips2011*. When you write the relative link that points to *antarctica.htm*, you need to include the name of the *Trips2011* subfolder, like this:

See pictures from Antarctica

This link gives the browser two instructions: first to go into the subfolder *Trips2011*, and then to get the page *antarctica.htm*. In the link, you separate the folder name (*Trips2011*) and the file name (*antarctica.htm*) with a slash character (/). Figure 8-3 shows both sides of this equation.

The figure consists of two screenshots of Microsoft Internet Explorer windows. The top window is titled 'Welcome' and has the URL 'file:///C:/Creating%20a%20Website/Chapter%208/TripRoot/index.htm'. It displays a page titled 'A Trip to Remember' featuring a large photo of a man sitting at a desk overwhelmed by papers, followed by a text block and several links: 'See pictures from Antarctica' (with a cursor arrow pointing to it), 'See pictures from China', 'See pictures from Idaho', 'About Me', and 'Email Me'. The bottom window is titled 'Antarctica' and has the URL 'file:///C:/Creating%20a%20Website/Chapter%208/TripRoot/Trip20'. It displays a grid of four images with captions: 'I fear penguins.', 'This polar bear claimed my brother.', 'Argh! My eyes sting.', and 'Colder than it looks.' Below the images are links: 'See what happened in China' and 'See what happened in Idaho'.

Figure 8-3: Using a relative link, you can jump from the main index.htm page (top) to a page with picture thumbnails (bottom). Each picture is itself a link—visitors click it to see a larger-sized version of the photo.

Interestingly, you can use relative paths in other HTML elements, too, like the `<style>` element and the `` element. For example, imagine you want to display the picture `photo01.jpg` on the page `index.htm`. This picture is two subfolders away, in a folder called Photos, which is inside Trips2011. But that doesn't stop you from pointing to it in your `` element:

```

```

Using this technique, you can dig even deeper into subfolders of subfolders of subfolders. All you need to do is add the folder name and a slash character for each subfolder, in order.

Remember, however, that relative links are always *relative to the current page*. If you want to display the same picture, `photo01.jpg`, on the `antarctica.htm` page, the `` element above won't work, because the `antarctica.htm` page is actually in the Trips2011 folder. (Take a look back at Figure 8-2 if you need a visual reminder of the site structure.) From the Trips2011 folder, you only need to go down one level, so you need this link:

```

```

By now, you've probably realized that the important detail lies not in how many folders you have on your site, but in how you organize the subfolders. Remember, a relative link always starts out from the *current* folder, and works its way up or down to the folder holding the target page.

Tip: Once you start specifying subfolders in relative links, you shouldn't change any of their names or move them around because you may break a link. That said, many web page editors (like Expression Web) are crafty enough to adjust your relative links. That's yet another reason to think about getting a full-featured web editor.

Moving up into a parent folder

The next challenge you face is going *up* a folder level. To do this, use the character sequence `..` (two periods and a slash). For example, to add a link in the `antarctica.htm` page that brings the reader back to the `index.htm` page, you write a link that looks like this:

```
Go <a href="../index.htm">back</a>
```

And as you've probably guessed by now, you can use this command twice in a row to jump up two levels. For example, if you have a page in the Photos folder that leads to the home page, you need this link to get back there:

```
Go <a href=".../../index.htm">back</a>
```

For a more interesting feat, you can combine both of these tricks to create a relative link that travels up one or more levels, and then travels down a different path. For example, you need this sort of link to jump from the `antarctica.htm` page in the Trips2011 folder to the `idaho.htm` page in the Trips2010 folder:

```
See what happened in <a href=".../Trips2010/idaho.htm">Idaho</a>
```

This link moves up one level to the root folder, and then back down one level to the Trips2010 folder. You follow the same process when you browse folders to find files on your computer.

Moving to the root folder

The only problem with the relative links you've seen so far is that they're difficult to maintain if you ever reorganize your site. For example, imagine you have a web page in the root directory of your site. Say you want to feature an image on that page that's stored in the *images* subfolder. You use this link:

```

```

But then, a little later on, you decide your page really belongs in *another* spot—a subfolder named Plant—so you move it there. The problem is that this relative link now points to *plant/images/flower.gif*, which doesn't exist—the Images folder isn't a subfolder in Plants, it's a subfolder in your site's root folder. As a result, your browser displays a broken link icon.

There are a few possible workarounds. In programs like Expression Web, when you drag a file to a new location, the HTML editor updates all the relative links automatically, saving you the hassle. Another approach is to try to keep related files in the same folder, so you always move them as a unit. However, there's a third approach, called *root-relative links*.

So far, the relative links you've seen have been *document-relative*, because you specify the location of the target page relative to the current document. *Root-relative* links point to a target page *relative to your website's root folder*.

Root-relative links always start with the slash (/) character (which indicates the root folder). Here's the `` element for *flower.gif* with a root-relative link:

```

```

The remarkable thing about this link is that it works no matter where you put the web page that contains it. For example, if you copy this page to the Plant subfolder, the link still works, because the first slash tells your browser to start at the root folder.

The only trick to using root-relative folders is that you need to keep the real root of your website in mind. When using a root-relative link, the browser follows a simple procedure to figure out where to go. First, it strips all the path and file name information out of the current page address, so that only the domain name is left. Then it adds the root-relative link to the end of the domain name. So if the link to *flower.gif* appears on this page:

```
http://www.jumboplants.com/horticulture/plants/annuals.htm
```

The browser strips away the */horticulture/plants/annuals.htm* portion, adds the relative link you supplied in the `src` attribute (*/images/flower.gif*), and looks for the picture here:

```
http://www.jumboplants.com/images/flower.gif
```

This makes perfect sense. But consider what happens if you don't have your own domain name. In such a case, your pages are probably stuck in a subfolder on another web server. Here's an example:

`http://www.superISP.com/~user9212/horticulture/plants/annuals.htm`

The domain name part of the URL is *www.superISP.com*, but for all practical purposes, the root of your website is your personal folder, *~user9212*. That means you need to add this detail into all your root-relative links. So to get the result you want with the *flower.gif* picture, you need to use this messier root-relative link:

``

Now the browser keeps just the domain name part of the URL (*www.super ISP.com*) and adds the relative part of the path, starting with your personal folder (*/~user9212*).

UP TO SPEED

The Rules for URLs

The rules for correctly writing a URL in an anchor element are fairly strict, and a few common mistakes creep into even the best web pages. Here are some pointers to help you avoid these headaches:

- When you create an absolute URL, you have to start with its protocol (usually `http://`). You don't need to follow this rule when typing a URL into a browser, however. For example, if you type *www.google.com*, most browsers are intelligent enough to assume the `http://` part. However, in an HTML document, it's mandatory.
- Don't mix up the backslash (`\`) and the ordinary forward slash (`/`). Windows uses the backslash in file paths (like `C:\Windows\win.ini`). In the web world, the forward slash separates subfolders (as in `http://www.ebay.com/Help/index.html`). Once again, many browsers tolerate backslash confusion, but the same mistake in an anchor element breaks your links.
- Don't ever use file paths instead of a URL. It's possible to create a URL that points to a file on your computer using the *file* protocol (as in `file:///C:/Temp/myPage.htm`). However, this link won't work on anyone else's computer, because they won't have the same file on their hard drive. Sometimes, design tools like Expression Web may insert one of these so-called local URLs (for example, if you drag and drop a picture file into your web page). Be vigilant—check all your links to make sure this doesn't happen.
- Don't use spaces or special characters in your file or folder names, even if these special characters are allowed. For example, it's perfectly acceptable to put a space in a file name (like *My Photos.htm*), but in order to request this page, the browser needs to translate the space into a special character code (*My%20Photos.htm*). To prevent this confusion, steer clear of anything that isn't a number, letter, dash (-), or underscore (_).

Changing Link Colors and Underlining

Virtually everyone born since 1900 instinctively understands that blue underlined text is there to be clicked. But what if blue links are at odds with the overall look of your site? Thanks to style sheets, you don't need to play by the link-color rules.

Based on what you learned about CSS in Chapter 6, you can quickly build a style sheet rule that changes the text color of all the link-producing anchor tags on your site. Here's an example:

```
a {  
    color: fuchsia;  
}
```

But watch out: custom link colors change the way the links behave. Ordinarily, when you click a link, it turns purplish red to show that you visited the page. Custom links, however, never change color—they retain their hue even after you click them.

A better way to create colorful links is to use another style sheet trick: *pseudo-classes*. Pseudo-classes are specialized versions of the CSS classes you learned about earlier (see page 147). They rely on details that a browser tracks behind the scenes. For example, ordinary classes apply rules indiscriminately to a given element, like an anchor tag. But pseudo-classes apply rules to elements that meet certain criteria, in this case links that are either clicked or unclicked.

Four pseudo-classes help you format links. They are *:link* for links that point to virgin ground; *:visited* for links a reader has already visited; *:active*, the color a link turns as a reader clicks it, before releasing the mouse button; and *:hover*, the color a link turns when a visitor moves his mouse over it. As you can see, pseudo-classes always start with a colon (:).

Here's a style rule that uses pseudo-classes to create a misleading page—one where visited links are blue and unvisited links are red:

```
a:link {  
    color: red;  
}  
a:visited {  
    color: blue;  
}
```

If you want to apply these rules to some, but not all, of your links, you can add a class name to your rule:

```
a.BackwardLink:link {  
    color: red;  
}  
a.BackwardLink:visited {  
    color: blue;  
}
```

Now an anchor element needs to specify the class name to display your new style, as shown here:

```
<a class="BackwardLink" href="...">>...</a>
```

Finally, it's worth noting that you can use this technique with the *text-decoration* style sheet property to change whether browsers automatically underline links. Here's an example that removes the standard underlining:

```
a {  
    text-decoration: none;  
}
```

This technique is generally a bad idea with links you embed in the main content of a page, because it can make them hard to spot. However, it's useful if you have a panel that consists of nothing but links (like a menu sidebar) and you want to give it a cleaner look.

mailto Links

A *mailto* link is a special type of link that helps visitors send a message to you. When you click a mailto link, your browser opens your email program and begins creating a message. It's still up to you to actually send the message, but the mailto link can get the process started with a boilerplate subject line and body text.

Note: The mailto link doesn't always work for visitors using web-based email services (like Hotmail and Gmail). Clicking a mailto link in a message may open a desktop email program they never use or even generate an error message (depending on their browser and computer settings.) To solve the problem or find a workaround on your computer, do a Google search for "mailto link" and your browser name.

To create a mailto link, specify a path that starts with the word "mailto," followed by a colon (:) and your email address. Here's an example:

```
<a href="mailto:me@myplace.com">Email Me</a>
```

Most browsers also let you supply text for the message's subject line and body. When someone clicks the mailto link, the new message loads this information, ready for sending (or editing).

To supply the subject line and body text, you have to use a slightly wonky syntax that follows these rules:

- Put a question mark after the email address.
- To declare the subject line, add `subject=` followed by the subject text.
- If you want to define some body text, add the character sequence & after your subject text, and then type `body=` followed by the body text.
- Replace characters that could cause problems with specialized codes. Letters, numbers, and the period are all fine, but most other punctuation isn't. For example, you have to replace every space in the subject and body text with the character sequence %20. This gets quite tedious and makes your message hard to read after you compose it, but it ensures that the mailto link works in every browser. The easiest way to prepare your message text is to visit a page like <http://meyerweb.com/eric/tools/dencoder>, which adds the code for you. Simply enter your message text into the provided text box, and then click Encode to replace potentially problematic codes with the appropriate ones.

Confused? The easiest way to grasp these rules is to take a look at a couple of examples. First, here's a mailto link that includes the subject text "Automatic Email":

```
<a href="mailto:me@myplace.com?subject=Automatic%20Email">  
Email Me</a>
```

And here's a link that includes both subject text and body text:

```
<a href=  
"mailto:me@myplace.com?subject=Automatic%20Email&body=  
I%20love%20your%20site.">Email Me</a>
```

When you click this link, you'll probably see some sort of warning message informing you that the web page is about to open your email program and asking your permission (the exact message depends on your browser and operating system). If you agree, you see an email form pop up, like the one shown in Figure 8-4.

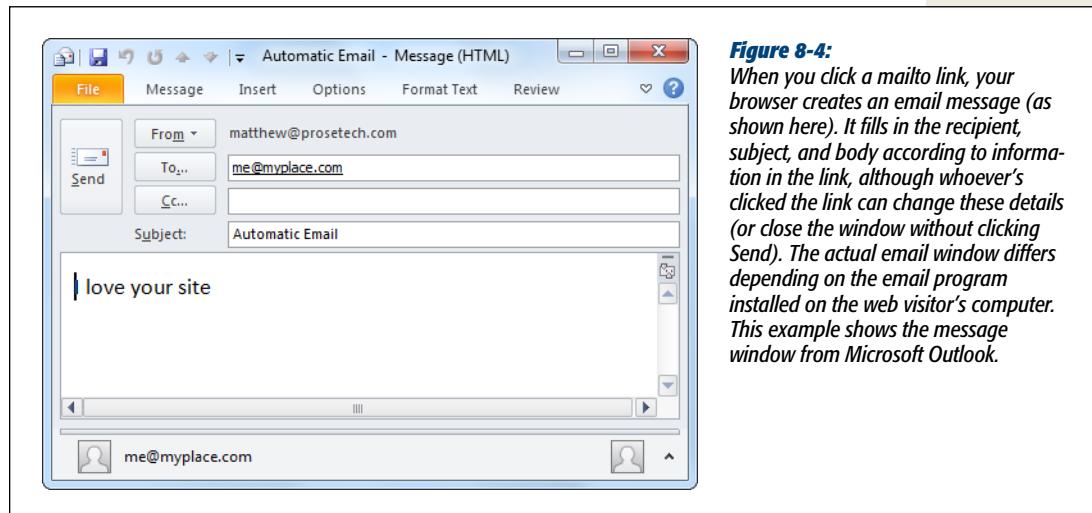


Figure 8-4:

When you click a mailto link, your browser creates an email message (as shown here). It fills in the recipient, subject, and body according to information in the link, although whoever's clicked the link can change these details (or close the window without clicking Send). The actual email window differs depending on the email program installed on the web visitor's computer. This example shows the message window from Microsoft Outlook.

Linking to Other Types of Content

Most of the links you write will point to bona fide HTML web pages. But that's not your only option. You can link directly to other types of files as well. The only catch is that it's up to the browser to decide what to do when someone clicks a link that points to a different type of file.

Here are some common examples:

- **You can link to a JPEG, GIF, or PNG image file.** When visitors click a link like this, the browser displays the image in a window without any other content. Websites often use this approach to let visitors take a close-up look at large graphics. For example, the Trips2011 website in the previous section has a page chock full of small image thumbnails. Click one of them, and the full-size image appears.
- **You can link to a specialized type of file, like a PDF file, a Microsoft Office document, or an audio MP3 file.** When you use this technique, you're

taking a bit of a risk. These links rely on a browser having a plug-in (a mini-program that handles specific tasks) that recognizes the file type or on your visitor having a suitable program installed on his computer. If the machine doesn't have the right software, the only thing your visitors will be able to do is download the file (see the next point), where it will sit like an inert binary blob. However, if a browser has the right plug-in, a small miracle happens, and the file opens up right inside the browser window.

- **You can link to a file you want others to download.** If a link points to a file of a specialized type and the browser doesn't have the proper plug-in, visitors get a choice: They can ignore the content altogether, open it using another program on their computer, or save it on their computer. This is a handy way to distribute large files (like a ZIP file featuring your personal philosophy of planetary motion).

Image Links and Image Maps

You can turn images into links, too. The trick is to put an `` element inside an anchor element (`<a>`), like this:

```
<a href="LinkedPage.htm"></a>
```

HTML adds a thick blue border around pictures to indicate that visitors can click them. Usually, you want to turn this chunky-looking border off using the style sheet border properties described on page 172. When a visitor points her cursor to a linked picture, the cursor changes to a hand.

Note: Text and images aren't the only thing you can put inside an anchor. In fact, anchors can accommodate any HTML, including entire paragraphs of text, bulleted lists, and so on. If you try this, you'll see that all the text inside becomes blue and underlined, and all the images inside sport blue borders. Web browsers have supported this bizarre behavior for years, but it's only HTML5 that makes it an official part of the HTML standard.

In some cases, you might want to create distinct clickable regions, called hotspots, *inside* a picture. For example, consider Figure 8-5.

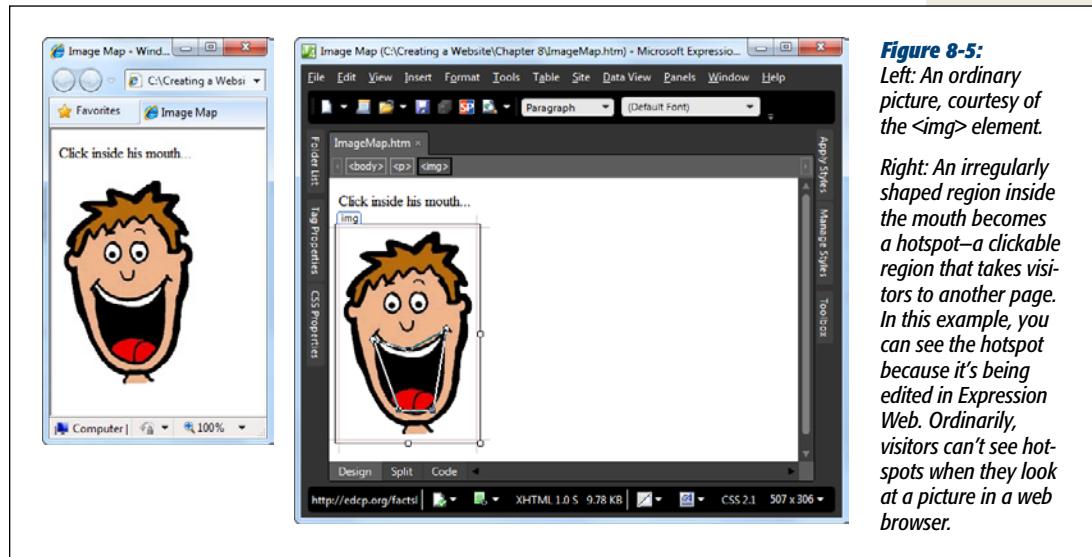


Figure 8-5:
Left: An ordinary picture, courtesy of the `` element.

Right: An irregularly shaped region inside the mouth becomes a hotspot—a clickable region that takes visitors to another page. In this example, you can see the hotspot because it's being edited in Expression Web. Ordinarily, visitors can't see hotspots when they look at a picture in a web browser.

To add a hotspot to a picture, you start by creating an *image map* using HTML's `<map>` element. This part's easy—all you do is choose a unique name for your image map so you can refer to it later on:

```
<map id="FaceMap" name="FaceMap">
</map>
```

Note: If you noticed that the `<map>` element uses two attributes that duplicate the same information (`id` and `name`), you're correct. Although in theory just the `id` attribute should do the trick, you need to keep the `name` attribute there to ensure compatibility with a wide range of browsers.

Then you need to define each hotspot, which you do inside the `<map>` element, between its start and end tags. You can add as many hotspots as you want, although they shouldn't overlap. (If they do, the one defined first takes precedence.)

To define each hotspot in an image, you add an `<area>` element. The `area` element identifies three important details: the target page a visitor goes to after clicking the hotspot (the `href` attribute), the shape of the hotspot (the `shape` attribute), and the exact dimensions of the shape (the `coords` attribute). Much like an image, the `<area>` element requires an `alt` attribute with some alternate text that describes the image map to search engines, reader programs, and ancient text-only browsers.

Here's a sample `<area>` element:

```
<area href="MyPage.htm" shape="rect" coords="5,5,95,195"
alt="A clickable rectangle" />
```

This hotspot defines a rectangular region. When visitors click it, they go to *MyPage.htm*.

The *shape* attribute supports three types of shape, each of which corresponds to a different value for the attribute. You can use circles (*circle*), rectangles (*rect*), and multi-edged shapes (*poly*). Once you choose your shape, you need to supply the coordinates, which are a bit trickier to interpret. To understand hotspot coordinates, you first need to understand how browsers measure pictures (see Figure 8-6).

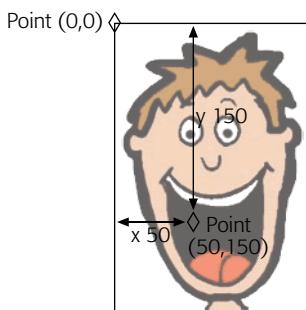


Figure 8-6:

Browsers designate the top-left corner of a picture as point (0, 0). As you move down the picture, the y-coordinate (the second number) gets bigger. For example, the point (0, 100) is at the left edge of the picture, 100 pixels from the top. As you move to the right, the x-coordinate gets bigger. That means the point (100, 0) is at the top of a picture, 100 pixels from the left edge.

You enter image map coordinates as a list of numbers separated by commas. For a circle, list the coordinates in this order: center point (x-coordinate), center point (y-coordinate), radius. For any other shape, supply the corners in order as a series of x-y coordinates, like this: x1, y1, x2, y2, and so on. For a polygon, you supply every point. For a rectangle, you only need two points: the top-left corner and the bottom-right corner.

You define the rectangle in the example above by these two points: (5, 5) at the top-left and (95, 195) at the bottom right. You define the more complex polygon that represents the mouth region in Figure 8-5 like this:

```
<area href="MyPage.htm" shape="poly" title="Smiling Mouth" alt="Mouth"  
coords="38, 122, 76, 132, 116, 110, 102, 198, 65, 197" />
```

In other words, your browser creates this shape by drawing lines between these five points: (38, 122), (76, 132), (116, 110), (102, 198), and (65, 197).

Note: Getting coordinates correct is tricky. Many web page editors, like Dreamweaver and Expression Web, have built-in hotspot editors that let you create an image map by dragging shapes over your picture, which is a lot easier than trying to guess the correct coordinates. To use this tool in Dreamweaver, select a picture, and then look for the three hotspot icons (circle, square, and polygon) in the Properties panel. Expression Web offers similar icons in the Picture toolbar. (If you can't see the Picture toolbar, right-click the picture, and then select Show Pictures Toolbar.)

Once you perfect all your hotspots, there's one step left: to apply the hotspots to the image by adding the *usemap* attribute to your ** element. Use the same name for this attribute as you did for the image map itself, but precede it with the number-sign character (#), which tells browsers that you defined an image map for the picture:

```

```

Here's the complete HTML for the mouth hotspot example:

```
<!DOCTYPE html>

<html>

<head>
<title>Image Map</title>
<style type="text/css">
img {
    border-style: none;
}
</style>
</head>

<body>
<p>Click inside his mouth...</p>
<p>
    <map id="FaceMap" name="FaceMap">
        <area href="http://edcp.org/factsheets/handfoot.html" shape="poly"
        coords="38, 122, 76, 132, 116, 110, 102, 198, 65, 197"
        alt="Smiling Mouth" />
    </map>
    
</p>
</body>

</html>
```

The hotspots you create are invisible (unless you draw lines on the picture to indicate where they are). When a visitor hovers over them, his mouse pointer changes to a hand. Clicking a hotspot has the same effect as clicking an ordinary `<a>` link: Your visitor gets transported to a new page.

Tip: It's tempting to use image maps to create links in all kinds of graphics, including buttons you may custom-design in an image editor. Hold off for a bit. Sophisticated websites like yours can go many steps further with menus and buttons, but to implement these nifty tricks you need the JavaScript know-how you'll learn in Chapters 15 and 16.

Adding Bookmarks

Most links lead from one page to another. When you make the jump to a new page, the browser plunks you down at the very top of the page. But you can also create links to *specific parts* of a page. This is particularly useful if you create long, scrolling pages and you want to direct your visitors' attention to a particular passage.

You can create links to another position on the *current* page (see Figure 8-7), or to a specific place in *another* web page. The place you send your reader is technically called a *fragment*.

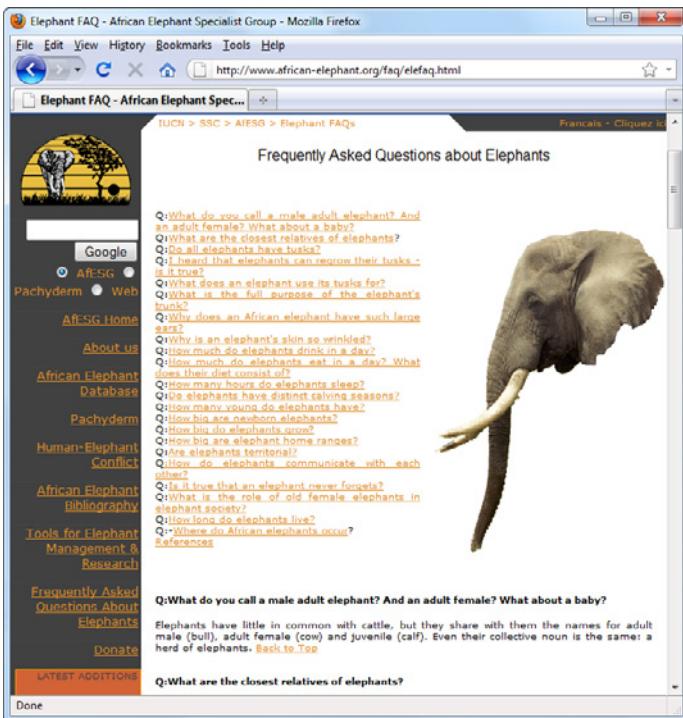


Figure 8-7:
This FAQ (frequently asked questions) page is an example of bookmarks at work. Here, the entire FAQ is a single long page, with a series of bookmark links at the top that let you jump to just the topic you're interested in. You could break an FAQ into separate pages, but readers wouldn't be able to scan through the whole list of questions, and they wouldn't have a way to print the entire document at once.

Creating a link that points to a fragment is a two-step process. First, you need to identify the fragment. You do this with the *id* attribute, which assigns a unique name to any HTML element on a page.

For example, imagine you want to send a visitor to the third level-3 heading in a web page named *sales.htm*. Initially, the markup looks like this:

```
...
<h3>Pet Canaries</h3>
<p>Pet canary sales have plummeted in the developed world, due in large part
to currency fluctuations and other macroeconomic forces.</p>
...
```

And here's the change that gives the Pet Canaries heading a unique name (in this case, the name is *Canaries*):

```
...
<h3 id="Canaries">Pet Canaries</h3>
<p>Pet canary sales have plummeted in the developed world, due in large part
to currency fluctuations and other macroeconomic forces.</p>
...
```

This doesn't affect the way the page looks; visitors never see the *id* attribute. However, it gives you a convenient way to drop visitors at the Pet Canaries heading. Essentially, you've created a bookmark that's locked on to this heading.

Once you create your bookmark, you can write a URL that points to it. The trick is to add the bookmark information to the end of the URL. To do this, you add the number-sign symbol (#) followed by the bookmark name.

For example, here's the link to send a reader to a bookmark named *Canaries* in the *sales.htm* page:

Learn about recent developments in canary sales.

When you click this link, the browser heads to the *sales.htm* page and scrolls down the page until it encounters the *Canaries* bookmark. The browser then displays, at the very top of the browser window, the text that starts with the heading *Pet Canaries*.

Tip: If your bookmark is near the bottom of a page, a browser might not be able to scroll the bookmark all the way to the top of its window. Instead, the bookmarked section appears somewhere in the middle of the browser window. This happens because the browser hits the bottom of the page, and can't scroll down any farther. If you suspect some potential for confusion (perhaps because you have several bookmarked sections close to each other at the bottom of a page), you can add a few
 elements at the end of your document, which lets the browser scroll down farther.

Sometimes you want to create a link that points to a bookmark in your *current* page. In this case, you don't need to specify a page name at all. Just start with the number sign, followed by the bookmark name:

Jump to the canary section.

Using bookmarks effectively is an art. Resist the urge to overcrowd your pages with links that direct readers to relatively small sections of content. Only use bookmarks to tame large pages that take several screensfull of scrolling.

When Good Links Go Bad

Now that you've learned all the ways to build links, it's a good time to consider what can go wrong. Links that go to pages on the same site can break when you rename or move files or folders. Links to other websites are particularly fragile; they can break at any time, without warning. You won't know that anything's gone wrong until you click the link and get a "Page Not Found" error message.

Broken links are so common that web developers have coined a term to describe how websites gradually lose their linking abilities: *link rot*. Sadly, you can upload a perfectly working website today and return a few months later to find that many of its external links have died off. They point to websites that no longer exist, have moved, or have been rearranged.

Link rot is an insidious problem because it reduces visitor confidence in your site. They see a page that promises to lead them to other interesting resources, but when they click a link, they're disappointed. Experienced visitors won't stay long at a site that's suffering from an advanced case of link rot—they'll assume you haven't updated your site in a while and move on to a snazzier site somewhere else.

So how can you reduce the problem of broken links? First, you should rigorously test all your internal links—the ones that point to pages within your own site. Check for minor errors that can stop a link from working, and travel every path at least once. Leading web page editors include built-in tools that automate this drudgery.

External links pose a different challenge. You can't create iron-clad external links, because link destinations are beyond your control and can change at any time. You could reduce the number of external links you include in your website to minimize the problem, but that isn't a very satisfying solution. Part of the beauty of the Web is the way a single click can take you from a comprehensive rock discography to a memorabilia site with hand-painted Elvis office supplies. As long as you want to connect your website to the rest of the world, you need to include external links. A better solution is to test your site regularly with a *link validator*, which walks through every one of your pages and checks each link to make sure it still leads somewhere.

In the following sections, you'll take a quick look at website management and link validators.

Site Management

Dreamweaver, Expression Web, and many other web page editors include site management tools that let you see your entire website at a glance. In most cases, you need to specifically define a site to take advantage of these features (a process described on page 92). Once you do, you get a bird's eye view of everything your site holds (see Figure 8-8).

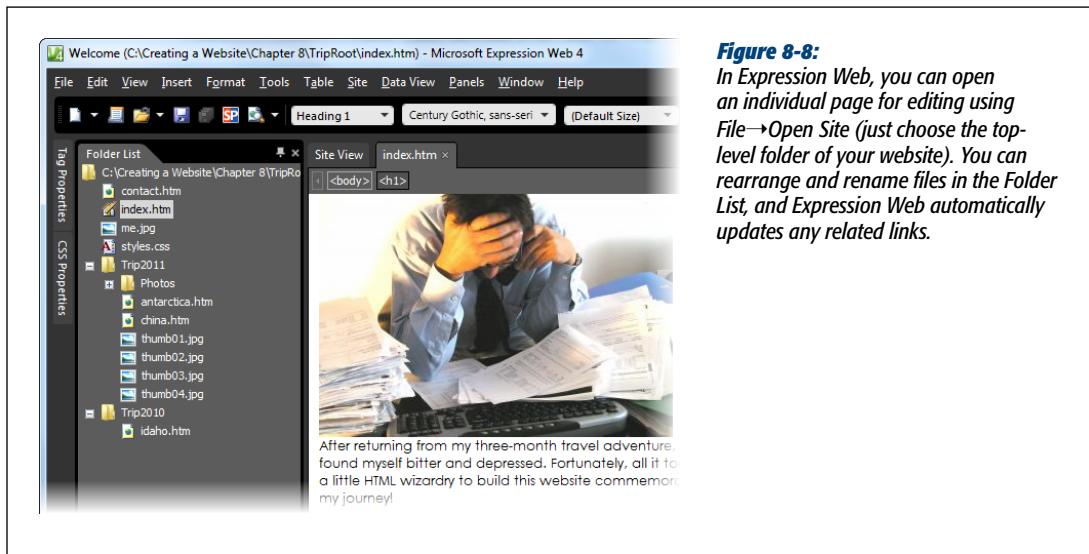
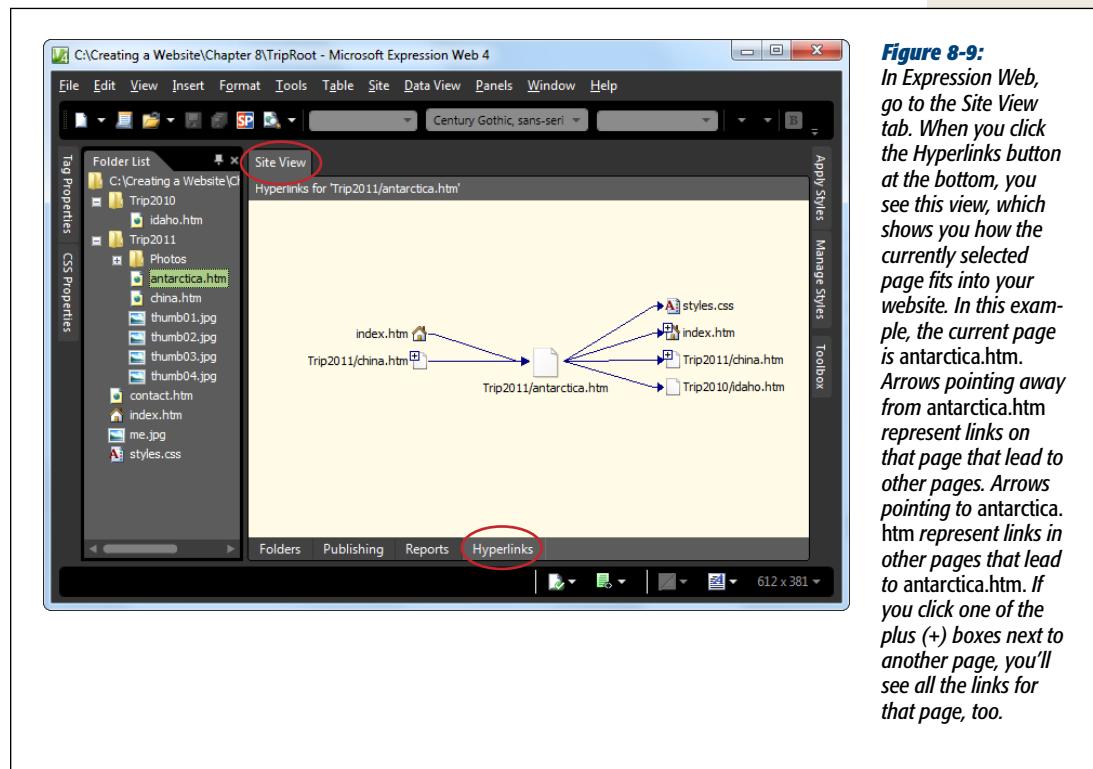


Figure 8-8:

In Expression Web, you can open an individual page for editing using File→Open Site (just choose the top-level folder of your website). You can rearrange and rename files in the Folder List, and Expression Web automatically updates any related links.

In many ways, looking at the contents of your website folders isn't as interesting as studying the web of links that binds your pages together. Many web page editors give you the ability to get an at-a-glance look at where all your links lead (see Figure 8-9).



Note: Expression Web's hyperlink viewer is one of the features that requires hidden metadata folders (page 93). If you open the hyperlink viewer and select a file but don't see any links, you probably haven't added the metadata folders yet. To get them, choose Site→Site Settings, turn on the "Maintain the website using hidden metadata files" setting, and then click OK.

Link Checkers

A *link checker* is an automated tool that scans one or more of your web pages. It tests each link it finds by trying to retrieve the target page (the page a link points to). Depending on the tool and the type of validation you're doing, link checkers might only scan internal links, or they might branch out to follow every link in every page until they've tested every link on your site.

Programs like Dreamweaver and Expression Web include sophisticated link checkers, and they're great for digging through your site and finding problems. In Dreamweaver, use the command Site→Check Links Sitewide to check links. In Expression Web, use a similar feature by choosing Site→Reports→Problems→Hyperlinks.

The link checkers built into these web page editors work on the copy of your website stored on your computer. That's the best way to keep watch for errors as you develop your site, but it's no help once your site's out in the wild. For example, it won't catch mistakes like a link to a file on your hard drive or to a file you forgot to upload to the web server.

To get the final word on your website's links, you might want to try a free online link checker. The World Wide Web Consortium provides a solid choice at <http://validator.w3.org/checklink>. To start your free online link check, follow these steps:

1. Go to <http://validator.w3.org/checklink>.

This takes you to the W3C Link Checker utility.

2. In the text box, enter the full URL of the page you want to check.

If your website has a default page like *index.htm*, you can type in just the domain name without explicitly supplying a file name.

3. Choose the options you want to apply (Figure 8-10).

Select "Summary only" if you want the checker to omit the detailed list of steps it takes as it examines each page. It's best to leave this option turned off so you can better understand exactly what pages the link checker examines.

Select "Hide redirects" if you want the checker to ignore instructions that would redirect it to a web page other than the target page specified (see page 236 for more). Usually, redirects indicate that your link still works, but also that you should update it to point to a new destination page.

The "Don't send the Accept-header" option prevents a link checker from telling a website its language preferences. This setting only matters if you're creating a multilingual website, which is beyond the scope of this book.

The "Check linked documents recursively" option validates links using recursion. If you don't use this option, the validator simply checks every link in the page you specify, and makes sure it points to a live web page. If you use recursion, the validator checks all the links in the current page, and then follows each internal link on your site. For example, if a link points to a page named *info.htm*, the link checker first verifies that *info.htm* exists. Then it finds all the internal links in *info.htm* and starts testing *them*. In fact, if *info.htm* links to yet another internal page (like *contact.htm*), the link checker branches out to that page and starts checking *its* links as well. The link checker is smart enough to avoid checking the same page twice, so it doesn't waste time checking links it has already validated.

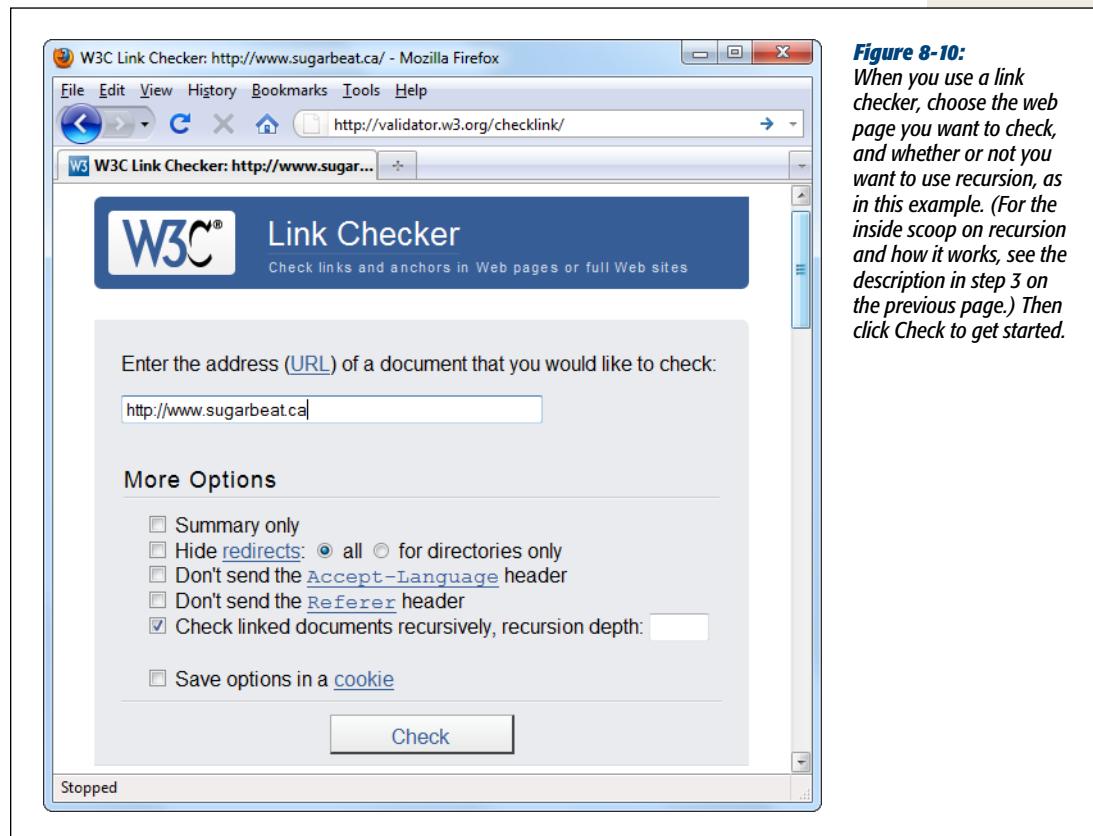


Figure 8-10:
When you use a link checker, choose the web page you want to check, and whether or not you want to use recursion, as in this example. (For the inside scoop on recursion and how it works, see the description in step 3 on the previous page.) Then click Check to get started.

Note: Link checkers don't use recursion on external links. That means that if you start your link checker on the home page of your website, it follows the links to get to every other page on your site, but won't go any further. Still, recursion's a great way to drill through all the links in your site in one go.

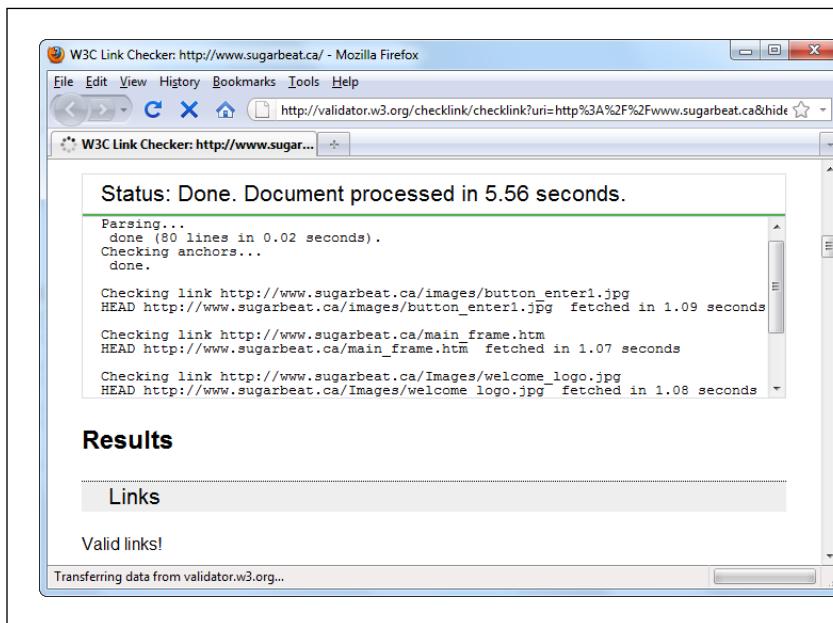
If you want to limit recursion (perhaps because you have a lot of pages and you don't need to check them all), you can supply a "recursion depth," which specifies the maximum number of levels the checker digs down. For example, with a recursion depth of 1, the checker follows only the first set of links it encounters. If you don't supply a recursion depth, the checker checks everything.

4. Select "Save options in a cookie" if you want your browser to remember the link-checker settings you specify.

If you use this option, the next time you use the link checker, your browser fills in the checkboxes using your previous settings.

5. Click “Check to start checking links.”

The link checker displays a report that lists each link it checks (Figure 8-11). It updates this report as it works. If you use recursion, you’ll see the link checker branch out from one page to another. The report adds a separate section for each page.



A screenshot of a Mozilla Firefox window titled "W3C Link Checker: http://www.sugarbeat.ca/ - Mozilla Firefox". The address bar shows the URL "http://validator.w3.org/checklink/checklink?uri=http%3A%2F%2Fwww.sugarbeat.ca&hide". The main content area displays the results of a link check. At the top, it says "Status: Done. Document processed in 5.56 seconds." Below this, there is a log of link checks:

```
Parsing...  
done (80 lines in 0.02 seconds).  
Checking anchors...  
done.  
  
Checking link http://www.sugarbeat.ca/images/button_enter1.jpg  
HEAD http://www.sugarbeat.ca/images/button_enter1.jpg fetched in 1.09 seconds  
  
Checking link http://www.sugarbeat.ca/main.frame.htm  
HEAD http://www.sugarbeat.ca/main.frame.htm fetched in 1.07 seconds  
  
Checking link http://www.sugarbeat.ca/Images/welcome_logo.jpg  
HEAD http://www.sugarbeat.ca/Images/welcome_logo.jpg fetched in 1.08 seconds
```

Below the log, there is a section titled "Results" with a "Links" heading. Under "Links", it says "Valid links!". At the bottom of the window, it says "Transferring data from validator.w3.org...".

Figure 8-11:
The link checker's final report shows a list of links found in anchors and images. The checker highlights links that lead to dead ends in red, and flags those that may need attention in yellow. One example of potential problem links are redirected links. Although they still work, they may be out of date and might not last for long.

Using Redirects

To be a good web citizen, you need to respect people who link to your site. That means that once you create your site and it becomes popular, try to avoid tinkering with page and folder names. Making a minor change could disrupt someone else’s link, making it difficult for return visitors to get back to your site.

Some web experts handle this problem using *redirects*. When they rearrange their sites, they keep all the old files, removing the content from them and replacing the old pages with a *redirect*—a special instruction that tells browsers to automatically navigate to a new page. The advantages of redirects are twofold: they prevent broken links, and they don’t lock you into the old structure of your site if you decide to make a change.

To create a redirect, you need to add a special <meta> element to the <head> portion of your web page. This element indicates the new destination using an absolute URL, and lists the number of seconds a browser should wait before performing the redirect. Here’s an example:

```
<!DOCTYPE html>

<html>

<head>
<meta http-equiv="REFRESH"
      content="10; URL=http://www.mysite.com/homepage.htm" />
<title>Redirect</title>
</head>

<body>
<h1>The page you want has moved</h1>
<p>
Please update your bookmarks. The new home page is
<a href="http://www.mysite.com/homepage.htm">
http://www.mysite.com/homepage.htm</a>.
</p>
<p>
You should be redirected to the new site in 10 seconds. Click
<a href="http://www.mysite.com/homepage.htm">
here</a> to visit the new page immediately.
</p>
</body>
</html>
```

To adapt this page for your own purposes, change the number of seconds (currently at 10) and the redirect URL. When a browser loads this page, it shows the temporary page for the indicated number of seconds, and then automatically requests the new page.

Redirected pages really serve two purposes: They keep your pages working when you change your site's structure, and they inform web visitors that the link is obsolete. That's where the time delay comes in—it provides a few seconds to notify visitors that they're entering the site the wrong way. Many sites keep their redirect pages around for a relatively short amount of time (for example, a year), after which they remove the page altogether.

Page Layout

In Chapter 6, you learned about the remarkable CSS standard, and how it can transform the drabbest of pages into a stylish gem. However, you still haven't seen everything style sheets let you do. Not only can you use them to play with colors, fonts, spacing, and borders, you can also use them to create sophisticated layouts. And with a little planning, style sheets let you create super-flexible pages that you can completely rearrange without touching a line of HTML. Instead, you simply swap in a new style sheet.

In this chapter, you'll see exactly how the CSS layout properties work. You'll learn to use modern layout techniques like floating boxes, side-by-side columns, and overlapping layers of pictures and text. But first, you need to take a step back to consider the challenges of layout on the Web and learn why online viewing isn't as straightforward as it seems.

The Challenge of Screen Space

When you design a page for print, you take into account the physical size of your final document. You'd use much larger text on a poster than on a business card, for example. But in the world of the Web, this system breaks down, because your website visitors can set their monitors to all kinds of screen resolutions, and resize their browser windows to all sorts of different dimensions. These details affect how much screen real estate your web pages have to work with. The higher the resolution and the bigger the browser window, the more of your content fits onscreen. This raises a dilemma—how do you make sure your pages look their best when you don't know your visitors' screen settings?

Web designers use two basic layout strategies to deal with this issue:

- **Go for flexibility with proportional sizing.** With proportional sizing, your layout expands or shrinks to fit the available space in a browser window. For example, if you create a proportionally sized web page with a fixed menu bar and a variable content area, the menu bar always stays at the same width, while the content area grows or contracts to fit the browser window, no matter how big or small your guest makes that window. If you're in doubt, proportional sizing is the way to go, because it ensures that your web pages will conform to any size browser window. However, you might want to impose some sort of maximum or minimum layout to prevent your pages from being scrambled beyond recognition. You'll learn how to do that in this chapter.
- **Pick a reasonable fixed width.** Sometimes, too much flexibility can cause its own problems. For example, if you shrink a proportionally sized page to extremely small dimensions, some page elements might get bumped into odd positions. If you have a complex layout with lots of graphics and floating elements, the result can be a bit of a mess. On the other hand, extremely large windows can cause problems, too. For example, if you stretch a proportionally sized page to fit the full width of a widescreen monitor, you might end up with extremely long lines of text that are hard to read. One solution is to use fixed-width pages that look good at a range of common browser settings.

Fixed-width sizing is the most common approach. Figure 9-1 shows it at work on the *New York Times* website (www.nytimes.com).

If you opt for the fixed-width solution, you need to figure out what that width should be. Of course, you have no way of knowing the most important factor affecting that decision—the size of your visitor's browser window. Recent browser statistics can help; they suggest that about 20 percent of people use relatively small displays that are 1,024 pixels wide. This includes some old-fashioned computer monitors, current netbooks, and the iPad. Virtually everyone else has a monitor with more pixels. Of course, as a web designer, you need to satisfy the lowest common denominator, so a fixed-width layout should look brilliant at a width of about 1,000 pixels. This way, web trekkers on small-screen devices can still enjoy all the content on your pages, provided they enlarge their browser window to fill the whole screen.

Tip: Some web page editors let you open pages in a range of browser window sizes. For example, in Expression Web you can choose File→Preview in Browser, which has options for a few standard, but old, window sizes. And some web browsers have add-ons that do the same thing. For example, Firefox and Chrome fans can use the Web Developer extension (<http://chrisspederick.com/work/web-developer>), which adds a toolbar packed full of handy web design tools, including an option to quickly change the size of the browser window.

**Figure 9-1:**

Top: To read this article without scrolling, you need a window that's about 600 pixels wide, which is a comfortable fit for an old 800 × 600 pixel monitor.

Bottom: Widen the window to about 1,000 pixels and you see frills like videos, ads, a search box, and a list of popular articles. If you make the window any wider, all you see is blank space on the right side. Most news sites use fixed-width page design to deal with variable window sizes.

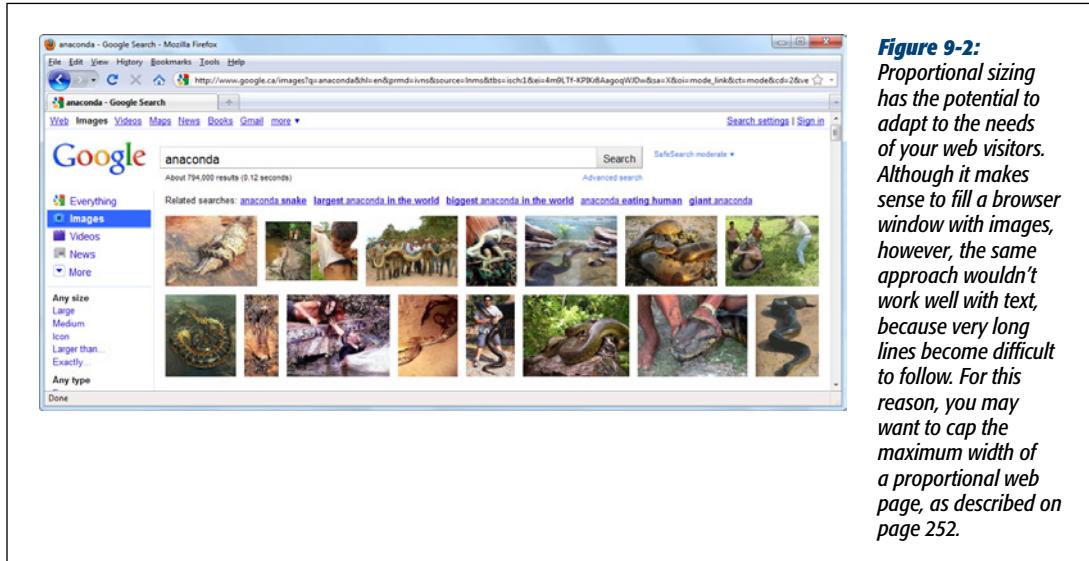
UP TO SPEED

Understanding Resolution

A pixel is the smallest unit of measurement on a monitor, and is otherwise known as a “dot.” For example, a resolution of 1024 × 768 means that a monitor displays a grid of pixels that’s 1,024 pixels wide and 768 pixels high (for a total of 786,432 pixels). The smallest resolution you’re likely to find today (outside of a mobile phone) is 1024 × 600, which are the dimensions of the typical netbook screen. But remember, people with large monitors won’t necessarily size their browser window to fill up the entire screen. (After all, it’s hard to read really long lines of text.) For that reason, 1,000 pixels is a good lower-limit assumption you can make for the width of a browser window.

To get some perspective, you might want to figure out what screen resolution you’re using—or even change it. To do so on Windows PCs, right-click the desktop, choose Personalize, click Display Settings, and then adjust the resolution using the handy slider. In earlier versions of Windows, you can find the same settings when you right-click the desktop and then select Properties→Settings. In Mac OS X, click System Preferences→Displays, and then select from the list of resolutions.

On some websites, proportional sizing works better than fixed-width layout, because it allows the website to make the best use of whatever screen space a visitor's got. For example, perform an image search on Google, and you'll notice that image thumbnails stretch to fill the whole width of the browser window, even on the widest widescreen monitor (Figure 9-2).



No matter which layout strategy you choose, you should test your pages at a variety of browser sizes to make sure your visitors see the best side of your work.

Finally, it's worth noting that this advice doesn't apply if you want to create a website for smartphones and other small mobile devices. Although all mobile devices can display ordinary websites, visitors will be forced to scroll, zoom in, and zoom out endlessly. To give mobile mavens the best experience, you need to use a mobile-only style sheet, and you probably need to rework the HTML of your web pages. For example, the *New York Times* provides a completely separate mobile website (<http://mobile.nytimes.com>). Mobile websites need to be compact, simple, and streamlined—after all, most users pay for every chunk of data they download on their phones.

Even though a huge number of people use web-enabled phones, the amount of web surfing done on mobile devices is still minuscule, and concentrated on just a few extremely popular websites (think Facebook and YouTube) and a few tasks (checking the weather, news, or sports scores).

Most website creators find that there's no reason to develop a mobile site unless it serves a specific need when people are out and about (like restaurant review or

movie-listing sites). For example, if you create a social site that turns out to be a blockbuster (say, the next Facebook), it's worth the effort to develop a full-fledged mobile site, but the effort is probably wasted on your personal blog. To learn more about mobile web development, check out the book *Programming the Mobile Web* (O'Reilly).

Style-Based Layout

In the early, lawless days of the Web, designers had to improvise their layout tools. One of their favorite tricks was the invisible table, which uses the `<table>` element you learned about in Chapter 5 to position rows and columns of content. Although invisible tables worked well enough, they left a tangled mess of markup in their wake. Today, web developers have largely given up on table-based layouts in favor of a cleaner and more powerful approach: style-based layout.

You've already taken your first tentative steps toward style-sheet nirvana by learning about style-based layout for boxed text and floating images in Chapter 7. You can apply that same idea to full pages. But before you go any further, it's a good idea to review the basic concepts that make style-based layouts possible.

Structuring Pages with the `<div>` Element

Before you start placing elements in specific positions on a page, you need a way to bundle related content into a single, neat package. In the table-based layout examples above, that package was the table cell. When you use style-based table layout, that package is the `<div>` element—the all-purpose container described on page 117.

Imagine you want to create a box with several links on the left side of your page. Positioning each link in that column is as much fun as peeling grapes. By using the `<div>` element, you can group everything together:

```
<div class="Menu">
  <a href="...">Home Page</a>
  <a href="...">Buy Our Products</a>
  <a href="...">File a Lawsuit</a>
  ...
</div>
```

Whenever you create a `<div>` element, you should choose a class name that describes the type of content it contains (like `Menu`, `Header`, `AdBar`, and so on). Later on, you can create a style rule that positions this `<div>` element and sets its font, colors, and borders.

Remember, a `<div>` element doesn't have any built-in formatting. In fact, on its own, it doesn't do anything at all. The magic happens when you combine your `<div>` element with a style sheet rule.

Floating Boxes

Ordinarily, HTML pages use a “flow” layout model; the content you provide flows from the top of the browser window to the bottom. The browser places elements one after the other, so they show up on the displayed page in the same order as they are in the HTML markup. When you use CSS layout properties, you take selected elements out of this system, and arrange them according to different rules.

One of the simplest layout techniques is to take a small portion of content and *float* it outside of the main layout of your page (see Figure 9-3). This way, the floating box sits wherever you place it, and the rest of the content flows around that box. In fact, you already used this technique to make pictures float in Chapter 7, using the style sheet property named *float*. A floating layout works just as readily with `<div>` elements as it did with those `` elements, with one exception: You also need to supply a width for the `<div>` element.

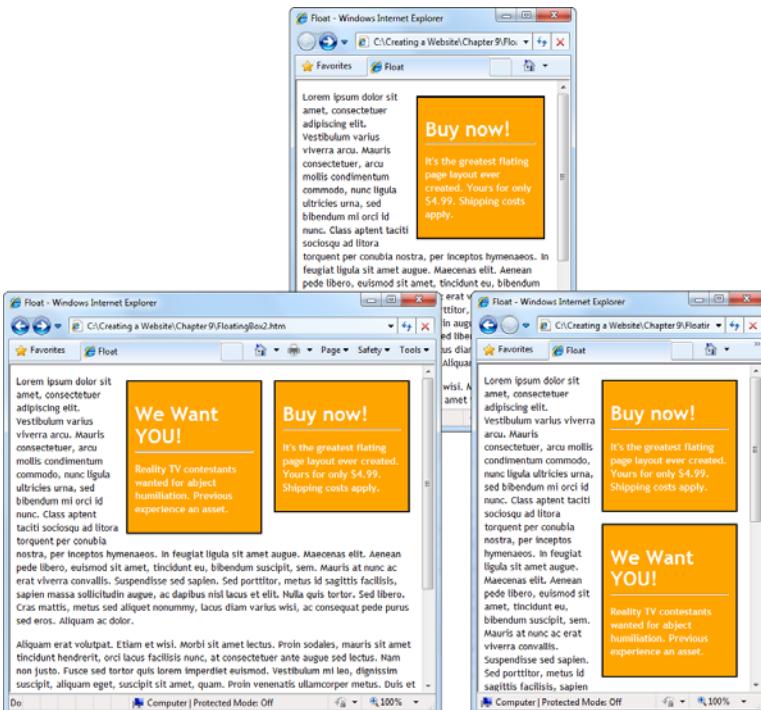
Note: When you float an image, browsers automatically make the floating box as wide as the image. When you float a `<div>` element with text inside, you can choose how wide you want it.

Figure 9-3:
Here are three examples
of floating layouts.

Top: A standard floating box.

Bottom, left: You can stack more than one floating box at a time. Your browser adds each new box to the left of the one before it.

Bottom, right: Add the `clear: both` style sheet property to force the second floating box to appear under the first.



Here's an example that defines a class that floats on the right side of some text:

```
.FloatingBox {  
    float: right;  
    width: 150px;  
    background-color: red;  
    border-width: 2px;  
    border-style: solid;  
    border-color: black;  
    padding: 10px;  
    margin: 8px;  
    font-weight: bold;  
    color: white;  
}
```

And here's the <div> that uses it:

```
<div class="FloatingBox">  
    <h1>Buy now!</h1>  
    <p>...</p>  
</div>
```

Note: The FloatingBox class sets the width, but not the height, of the box. That way, the box is made just big enough to fit the content inside. You could specify a fixed height using the *height* property, but you might truncate the end of your text (if it's too small) or leave extra white space at the bottom (if it's too big).

Fixed Boxes

The examples in Figure 9-3 are called floating boxes because they can “float” around the page to different positions, depending on the size of the browser window and the content that surrounds them. When a browser encounters a <div> element that uses the *float* property, it positions that element on the side of the page you specify (left or right). It positions the top of the <div> at the point on the page where it encounters the <div> element in the HTML. So if the browser finds a <div> halfway down the page it's creating, it puts the floating box halfway down the page.

Style sheets give you another option: You can place an element in a set, unchanging position. To do so, use the positional properties *top*, *left*, *bottom*, *right* in conjunction with the *position* property (set to *absolute* in this case). Here's an example:

```
.FixedBox {  
    position: absolute;  
    top: 20px;  
    left: 0px;  
    width: 150px;  
    background-color: orange;  
    border-width: 2px;  
    border-style: solid;  
    border-color: black;  
    padding: 10px;  
    margin: 8px;
```

```
font-weight: bold;  
color: white;  
}
```

Unlike floating boxes (which float at the sides of a page), fixed boxes can go anywhere. When you specify the location of an element using absolute positioning, you remove that element from the normal “flow” of the page. As a result, the rest of your content won’t wrap around a fixed box. Instead, the fixed box sits on top of the content, as you can see in Figure 9-4.

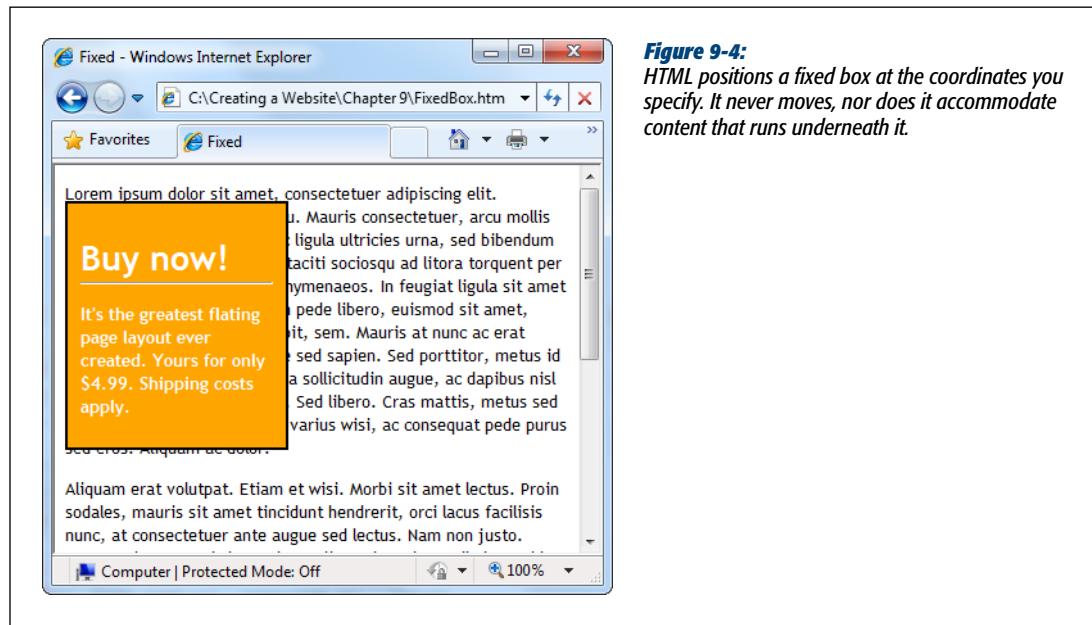


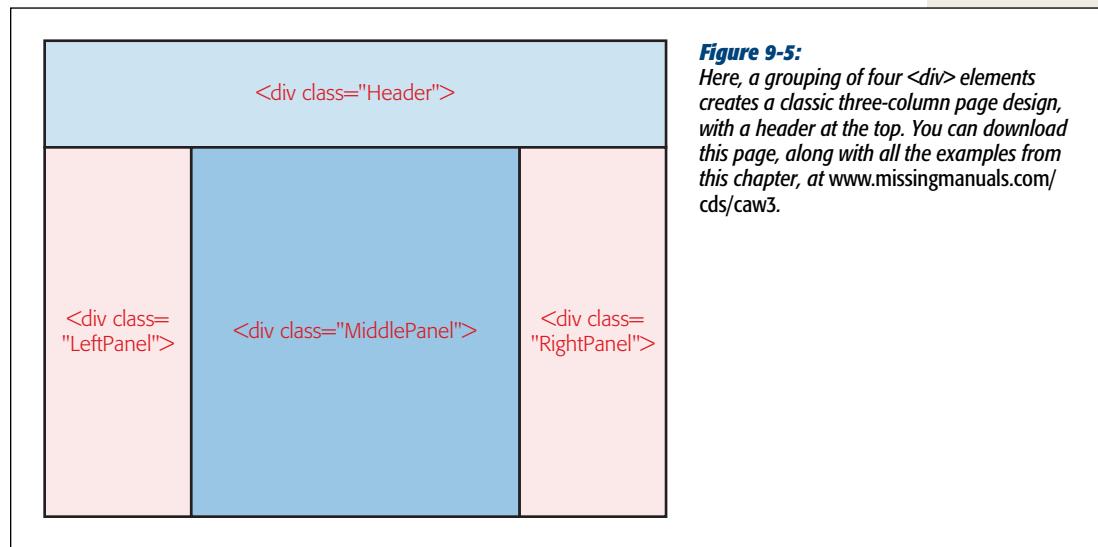
Figure 9-4:

HTML positions a fixed box at the coordinates you specify. It never moves, nor does it accommodate content that runs underneath it.

At first glance, fixed boxes seem like a problem. After all, who wants to deal with a jumble of overlapping text? However, fixed layout is a key ingredient in the multicolumn layouts you’ll learn about next.

Creating a Layout with Multiple Columns

As you’ve seen, style sheets let you place elements at fixed locations on a page, with no wrapping involved. Used carelessly, this leads to scrambled pages. But used carefully, it lets you create some of the most common and classic web page layouts. The trick is to create *several* boxes, some for text and others for images, and position the boxes next to each other on the page (see Figure 9-5). That way, text and pictures will remain carefully and rigidly separated.

**Figure 9-5:**

Here, a grouping of four <div> elements creates a classic three-column page design, with a header at the top. You can download this page, along with all the examples from this chapter, at www.missingmanuals.com/cds/caw3.

One of the most common website designs is to divide a page into two or three columns. The column on the left typically holds navigation buttons or other links. The column in the middle takes up the most space and includes the main content for the page. The column on the right, if present, displays additional information, like an advertisement or another set of links.

Building this example is surprisingly easy. But before you can position the <div> elements that shape the page, you need to decide whether you want a fixed-width layout or a resizable one. (Flip back to page 240 for a summary of the differences.) In the following sections, you'll learn to create both.

Building a Fixed-Width Layout

To set up a fixed-width layout, you simply decide on the exact pixel width of each <div> element. For example, you might make the left and right panels 100 pixels wide, and the middle panel 450 pixels wide. This adds up (with a bit of margin space in between), to a total of about 700 pixels.

Tip: If you go with a fixed layout, follow the 1,000-pixel guideline and make sure all your content fits comfortably at this size, with no side scrolling required. The *New York Times* article (Figure 9-1) provides a good model. The 700-pixel-wide example in this section meets this guideline, but is a bit on the small side.

Once you decide how to allocate your space, you simply need to create three style rules, one for each column. As in the examples you saw earlier, you control the placement of the column by setting the left coordinate, and you let the content determine its total height.

The following style rule defines a panel that's 100 pixels wide and positioned along the left side of a page:

```
.LeftPanel {  
    position: absolute;  
    left: 0px;  
    width: 100px;  
}
```

The middle panel starts at the 120-pixel mark and fills up another 450 pixels. This leaves 20 pixels of blank space between the right side of the left panel and the left side of the middle panel. The middle panel also separates itself from the content on the side using border properties (page 172).

```
.MiddlePanel {  
    position: absolute;  
    left: 120px;  
    width: 450px;  
    border-left-width: 1px;  
    border-right-width: 1px;  
    border-top-width: 0px;  
    border-bottom-width: 0px;  
    border-style: solid;  
    border-color: blue;  
    padding-top: 0px;  
}
```

Finally, the panel on the right side starts at the 610-pixel mark and takes up 100 pixels.

```
.RightPanel {  
    position: absolute;  
    left: 610px;  
    width: 100px;  
    font-weight: bold;  
}
```

Although you could use absolute positioning for the `<div>` that holds the header at the top of the page, you don't need to take that step. Instead, just make sure that you place the header before the other `<div>` elements:

```
<div class="Header">  
    <h1>The Joy Of Styles</h1>  
</div>  
  
<div class="LeftPanel">  
    <a href="...">Page 1</a><br />  
    <a href="...">Page 2</a><br />  
    ...  
</div>  
  
<div class="RightPanel">  
    <p>Donate to my untraceable Swiss Bank account ...</p>  
</div>  
  
<div class="MiddlePanel">  
    <p>An expandable middle might not be ...</p>  
</div>
```

In the style sheet, don't specify the top coordinate for any of the <div> elements. This way, the browser will display the header, and then create the <div> elements at the current location, so that the top of each panel starts just underneath the header. (Another option is to manually set the width property for the header, and the top property for the two side panels, to explicitly set the horizontal placement of each section.)

You can see the end result in Figure 9-6.

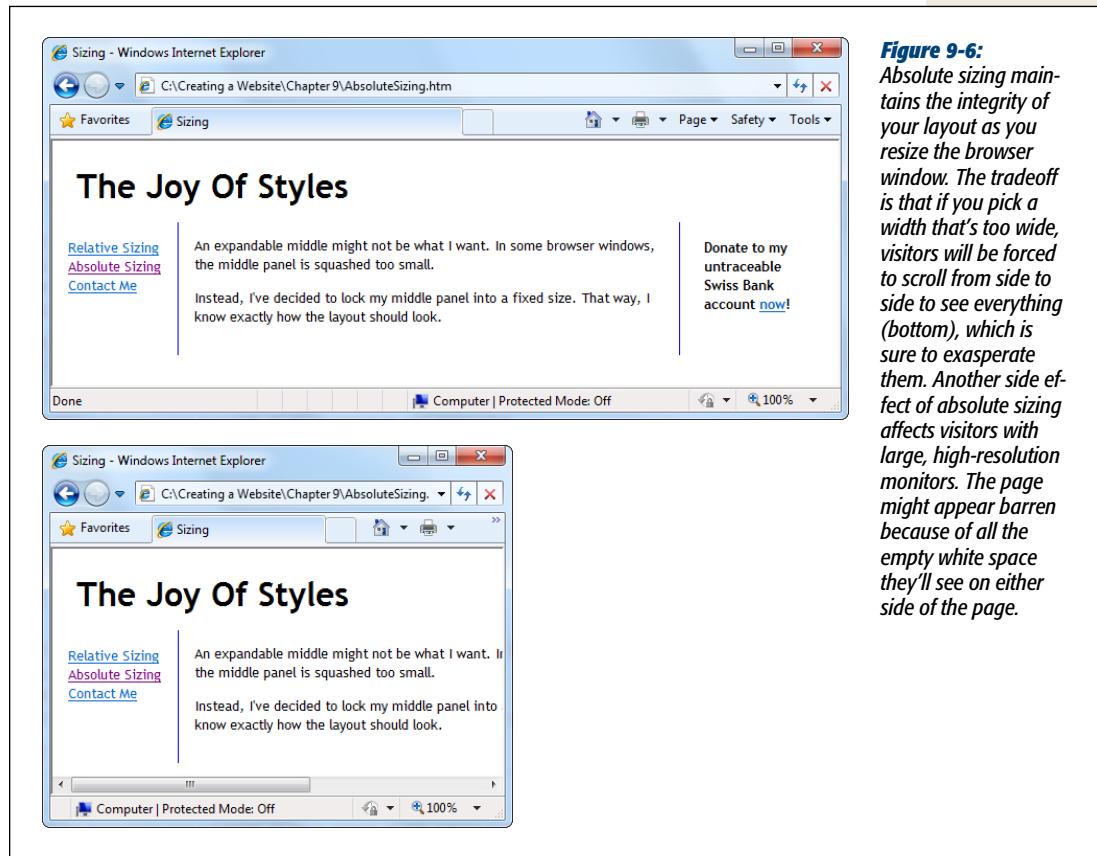


Figure 9-6:
Absolute sizing maintains the integrity of your layout as you resize the browser window. The tradeoff is that if you pick a width that's too wide, visitors will be forced to scroll from side to side to see everything (bottom), which is sure to exasperate them. Another side effect of absolute sizing affects visitors with large, high-resolution monitors. The page might appear barren because of all the empty white space they'll see on either side of the page.

The remarkable part about this example is that your HTML document is free of messy formatting details. Instead, it's a small miracle of clarity, with content divided into several easy-to-understand sections. And if you save your styles as an external style sheet (page 136), you can start building a second page using the same layout without spending any time puzzling out the correct formatting.

DESIGN TIME

Centering a Fixed-Width Layout

Sometimes, webmasters center fixed-width layouts horizontally. That way, their content always appears in the center of the browser window (rather than smushed up against the left edge).

Implementing this design is fairly easy in theory, but it can be a bit tricky in practice. First, you need to wrap your entire layout in another `<div>` element. In this example, it's named Body:

```
<div class="Body">
  <div class="Header">...</div>
  <div class="Left">...</div>
  <div class="Middle">...</div>
  <div class="Right">...</div>
</div>
```

You then need to set the width of this `<div>` to the total width of your full layout, and you need to set the right and left margins to `auto`:

```
.Body {
  width: 750px;
  margin-left: auto;
  margin-right: auto;
}
```

When you set auto margins on both sides, the browser automatically makes them equal. The result is that your element ends up centered in the middle.

However, the `<div>` wrapper trick isn't quite enough to solve the centering problem. When you use this technique, none of the `<div>` elements for the columns are allowed to use absolute positioning. The workaround is to float all the `<div>` columns on the left, one after the other. As long as you set a fixed width for each column and add them in the right order, this technique works without a hitch. To see it in practice, check out the *AbsoluteSizing_Centered.htm* file with the examples for this chapter at www.missingmanuals.com/cds/caw3.

Creating a Resizable Layout

A resizable layout is one in which the middle column—the one with the main content of your page—expands to take advantage of the available browser space. Figure 9-7 shows the difference.

You can adapt the style sheet used in the previous example (for a fixed layout) into one that applies a resizable layout. Interestingly, you don't need to change the style for the left panel. It can still use absolute positioning to lock onto the side:

```
.LeftPanel {
  position: absolute;
  left: 0px;
  width: 100px;
}
```

The panel on the right is slightly different. Because you don't know how wide the middle section will be, you don't know the left coordinate. You get around this in a tricky way: by using the `right` position property. This positions the `<div>` in relation to the `right` side of the browser window, so `0` is right against the right edge, `100px` is 100 pixels to the left, and so on. In this example, you give the panel 10 pixels of extra space to make sure its content doesn't rub up against the border of the browser window:

```
.RightPanel {  
    position: absolute;  
    right: 10px;  
    width: 100px;  
    font-weight: bold;  
}
```

The figure consists of two side-by-side screenshots of a Windows Internet Explorer window displaying a web page titled "The Joy Of Styles".

Top Screenshot: The browser window is relatively small. The layout features three columns: a narrow left column containing navigation links ("Relative Sizing", "Absolute Sizing", "Contact Me"), a narrow right column containing a call-to-action ("Donate to my untraceable Swiss Bank account [now!](#)"), and a wide middle column containing a paragraph of text about creating an expandable middle column in an HTML table.

Bottom Screenshot: The browser window is larger. In this view, the left and right columns remain the same width as in the top screenshot, but the middle column has resized to occupy the available horizontal space, making it wider than in the top screenshot.

Figure 9-7:
Top: At small sizes,
this layout looks the
same as its fixed-
width counterpart.

Bottom: Shrink or
stretch the window a
bit, and you'll see the
difference. The side
panels stay the same
size, but the middle
panel resizes itself
to fit.

The final step is to define the content section that sits between these two panels. You can't use absolute positioning for this, because you don't know how wide the browser window will be. Fortunately, there's another trick you can use—margin space. To make this work, you pretend that your middle panel has the full run of the browser window, like any normal piece of content. However, you pad the left and right margins with enough space to leave room for the side panels.

In this example, the left panel measures 100 pixels wide. Add 20 pixels of space in between, and that means your center panel needs a left margin of 120 pixels. The right margin is 130 pixels, because it needs 10 extra pixels to compensate for the space you left between the panel and the right edge of the browser window. Here's the final style sheet rule:

```
.MiddlePanel {  
    margin-left: 120px;  
    margin-right: 130px;  
    border-left-width: 1px;  
    border-right-width: 1px;  
    border-top-width: 0px;  
    border-bottom-width: 0px;  
    border-style: solid;  
    border-color: blue;  
    padding-top: 0px;  
}
```

There's one potential quirk in this example. In your markup, you now need to place the `<div>` for the left and right panels *before* the `<div>` for the middle panel. Otherwise, the browser will place the side panels after the main content, at the bottom of the page.

Tip: If it hurts your brain to think about horizontal positioning, there's an easy solution. Use the *top* position property to put each `<div>` in exactly the right spot. This makes your style sheet slightly longer, but it also ensures that you can place your `<div>` elements in any order, because you've told the browser exactly where they belong.

Maximum Width: The Safety Net

When you use resizable layouts, even the most respectable pages can run into two types of trouble. The first occurs if your visitor shrinks her browser window to ridiculously narrow dimensions. Space becomes so constricted that even a single word can't fit in the middle column, and the separate sections begin to overlap each other.

This isn't a huge problem—after all, most people don't expect a website to keep looking pretty when squashed paper-thin—but it's still a bit short of a professional page. The second problem rears its head if a visitor expands her browser window to fill all the space on a widescreen, high-resolution monitor. In that case, the middle panel is so wide that the text fits on just two or three lines. Besides looking odd, it's extraordinarily difficult to read.

You can prevent both these problems using two more CSS properties: *max-width* and *min-width*. The *max-width* property sets a maximum width beyond which an element will not expand. The *min-width* property sets a minimum width beyond which an element will not shrink. Essentially, when you hit these limits, your page turns into a fixed layout. Expand the page further than the maximum, and you get extra white space. Shrink it smaller than the minimum, and the browser gives you scrollbars.

You might think that you can put this technique into practice by applying *max-width* and *min-width* to the middle panel only. And you can—but the result won't be exactly what you want. If you limit the growth of the middle panel, the right panel will still follow the right edge of the browser, and gradually move farther and farther away from the content. The solution is to wrap the entire page, with all its panels, in another `<div>` container, as shown here:

```
<!DOCTYPE html>
<html>

<head>
  ...
</head>

<body>
  <div class="BodyContainer">
    <div class="Header">...</div>
    <div class="LeftPanel">...</div>
    <div class="RightPanel">...</div>
    <div class="MiddlePanel">...</div>
  </div>
</body>
</html>
```

Then you can set maximum and minimum size rules for this `<div>`:

```
.BodyContainer {
  position: absolute;
  max-width: 1000px;
  min-width: 100px;
}
```

This creates a perfect compromise between resizable and fixed-width sizing. Now, the middle column adapts to the best possible width within reasonable constraints, and the page looks professional no matter the size of the browser window.

Tip: You can try this example out in real life by downloading this chapter's companion content. It's available at www.missingmanuals.com/cds/caw3.

Stretching Column Height

So far, you've focused on making columns the right width, but you haven't worried much about height. After all, the browser takes care of that, by making an element just big enough to fit its content, along with any extra padding you specified with the *padding* property (page 153).

However, there are several situations where you need to size a column based on the size of the browser window rather than on the content it contains. For example, you might need to apply a background color that fills an entire column, not just the portion with text in it. Or, you might want to use borders that stretch to the bottom of the window (unlike the borders in the examples you've seen so far, which end with the text). Figure 9-8 shows this effect in practice.

Fortunately, you can dictate height using HTML's *min-height* property. The trick is to use a *percentage* size instead of a pixel size. For example, if you set *min-height* to 100 percent, the column will stretch to fill the browser window even if you have only a small amount of content.

Here's the formatting rule for the side panels in Figure 9-8:

```
.LeftPanel {  
    position: fixed;  
    top: 0px;  
    left: 0px;  
    left-padding: 10px;  
    width: 150px;  
    min-height: 100%;  
    background-color:#eee;  
    border-right-width: 1px;  
    border-right-style: solid;  
    border-right-color: black;  
    padding-bottom: 8px;  
}  
  
.RightPanel {  
    position: fixed;  
    top: 0px;  
    right: 0px;  
    width: 150px;  
    min-height: 100%;  
    background-color:#eee;  
    border-left-width: 1px;  
    border-left-style: solid;  
    border-left-color: black;  
}
```

The figure consists of two side-by-side screenshots of a Windows Internet Explorer browser window. Both screenshots show a web page titled "The Joy Of Styles". The page has a header, a left sidebar with links, a central content area, and a right sidebar with contact information.

Top Screenshot: This shows a standard three-panel layout. The left sidebar has a light gray background and a thin black border. The central content area and right sidebar both have white backgrounds and thin black borders. The text within these panels is clearly legible against the background.

Bottom Screenshot: This shows a variation of the same design. The left sidebar now has a black background and a thick black border, which appears to be 100% of the browser's height. The central content area and right sidebar also have black backgrounds and thick black borders. The text is less distinct against the dark background compared to the top version.

It's important to note that this example includes a slight change to the *position* property. Before, you set it to *absolute*, which places an element on a page using absolute coordinates. A value of *fixed* places the element using absolute coordinates, relative to the *viewport*—that's the window of content your browser displays on a single screen. In many cases, *absolute* and *fixed* have the same effect, but in this situation the distinction is important. That's because you want to size the panels relative to the size of the browser window, not relative to the size of the entire web page. If you use absolute positioning, the side panels might grow larger than necessary, forcing the browser to show scrollbars.

Super-Flexible Sites: The Zen of Web Design

By now, you might have the impression that style sheets are at least as important to a modern web page as HTML markup. In Chapter 6, you learned to use their formatting muscle. In this chapter, you've seen how to use style sheets to control complex layouts. Combine your newly learned skills, and you'll see that style sheets give you the ability to completely control the display of a well-designed web page.

Style sheets not only let you apply formatting and layout to a page, they also let you *change* the formatting and layout in one (immensely gratifying) step. This is the holy grail of web design—a way to update, revamp, and customize an entire site as your needs change. Figure 9-9 shows the CSS Zen Garden website, which fulfills this dream.

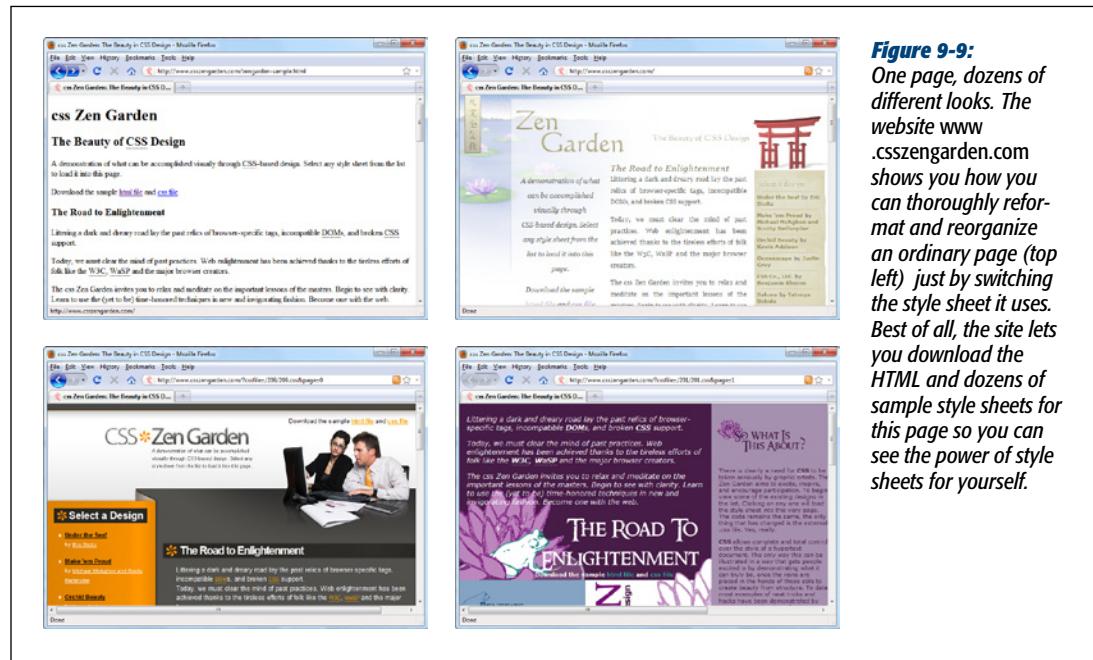


Figure 9-9:
One page, dozens of different looks. The website www.csszengarden.com shows you how you can thoroughly reformat and reorganize an ordinary page (top left) just by switching the style sheet it uses. Best of all, the site lets you download the HTML and dozens of sample style sheets for this page so you can see the power of style sheets for yourself.

Here's a quick recap of the design steps you need to follow to make this work on your website:

1. Plan your pages before you write a single tag.
Begin by making the layout of a page a collection of separate regions.
2. Put each region into a separate `<div>` element.
3. Give each `<div>` element a unique class name that reflects its purpose, not its format.

Do this for every section of a page, even if you don't intend to apply style sheet rules to it right now.

4. Finally, write the style sheet rules that position and format each <div> element.

This is the most time-consuming part of your markup to write, but it's time well spent. You can tweak your formatting rules any time without disturbing your content.

Tip: Planning to take your style sheet layouts to the next level? Check out the legendary books by Eric Meyer, like *Smashing CSS: Professional Techniques for Modern Layout* (Wiley).

POWER USERS' CLINIC

Advanced Style Sheet Layout

You've covered quite a bit of ground on your mission to build multicolumn layouts. However, web designers have spent years playing with CSS to create even more fresh, eye-popping designs. If you want to go beyond the standard fixed-width and resizable layout techniques you've learned so far, here are three new horizons to conquer:

- **Sticky Footers.** What if you could have a footer that doesn't sit at the bottom of your page, but at the bottom of your browser window, just under your columns and always accessible? Creating the markup for this on your own risks serious headaches, but there's a great free solution at www.cssstickyfooter.com. And like all worthwhile style sheet designs, it supports every browser.
- **Fluid Layouts.** Fluid layouts (also called liquid layouts) are similar to the relative layouts you used earlier in this chapter, except that they have several resizable columns instead of just one resizable middle panel. You control how a browser divvies up extra space among these columns by giving each column a percentage width. For example, if one column has a 50 percent width and two others have 25 percent widths, the first column always occupies half the browser window. In the right hands, fluid layouts can

create a range of dynamic effects. (For an advanced example, see www.dccdesign.co.uk.) However, it's sometimes difficult to anticipate the effects of resizing several columns at once, and you still need to set the *max-width* property (page 252) to prevent content from growing too wide and becoming difficult to read.

- **Elastic Layouts.** Elastic layouts are similar to fluid layouts, but you set the width of each column using the proportional *em* unit rather than a percentage. The difference is that the *em* unit is relative to the text size setting in the browser. This means that if a guest expands the default text size (for example, using the Page→Text Size menu in Internet Explorer), your entire layout expands to compensate, widening its columns to fit the enlarged text. You can see an example of this technique with CSS Zen Garden at <http://tinyurl.com/67v6hb6>.

For more information about fluid and elastic layout techniques, you can read a good article from Smashing Magazine at <http://tinyurl.com/q54tnl>. And if you own Dreamweaver, you can play with a range of ready-made fluid and elastic layouts. Just create a new page and choose one of the many templates.

A Few More Layout Techniques

Now that you've taken your first steps to becoming a style sheet layout guru, it's time to cover a few more techniques you might need to know. You won't use these as often as multicolumn layouts, but they're still good to keep in your back pocket.

Layering

Remember how you need to position elements carefully when you use absolute positioning to make sure you don't overlap one element with another? Interestingly, advanced web pages sometimes *deliberately* overlap elements to create dramatic effects. For example, you might create a logo by overlapping two words, or create a heading by partially overlapping a picture. These tricks use overlapping *layers*.

To use overlapping layers, you need to tell your browser which element goes on top. You do this through a simple number called the *z-index*. Browsers put elements with a high *z-index* in front of elements with a lower one.

For example, here are two elements positioned absolutely so that they overlap:

```
.Back {  
    z-index: 0;  
    position: absolute;  
    top: 10px;  
    left: 10px;  
    width: 150px;  
    height: 100px;  
    background-color: orange;  
    border-style: dotted;  
    border-width: 1px;  
}  
  
.Front {  
    z-index: 1;  
    position: absolute;  
    top: 50px;  
    left: 50px;  
    width: 230px;  
    height: 180px;  
    font: xx-large;  
    border-style: dotted;  
    border-width: 1px;  
}
```

The first class (Back) defines an orange background square. The second class (Front) defines a large font for text. You set the elements' *z-indexes* so that the browser superimposes the Front box (which has a *z-index* of 1) over the Back box (which has a *z-index* of 0). In the example above, the HTML adds a dotted border around both elements to make it easier to see how the boxes overlap on a page.

Note: The actual value for *z-index* isn't important, the only important characteristic is how it compares to others. For example, if you have two elements with *z-index* settings of 48 and 100, you'll get the same effect as two elements with values of 0 and 1: The second element overlaps the first. If two or more elements have the same *z-index* value, the one that's first in the HTML gets shoved underneath those that come later.

In your HTML, you need to create both boxes using `<div>` elements. It also makes sense to supply some text for the Front box:

```
<div class="Back">  
</div>  
  
<div class="Front">  
    This text is on top.  
</div>
```

Load this page in a browser and you'll see a block of text that stretches over part of the orange box and out into empty space (see Figure 9-10, left).

You can reverse the *z-index* to change the example:

```
.Back {  
    z-index: 1;  
    ...  
}  
.Front {  
    z-index: 0;  
    ...  
}
```

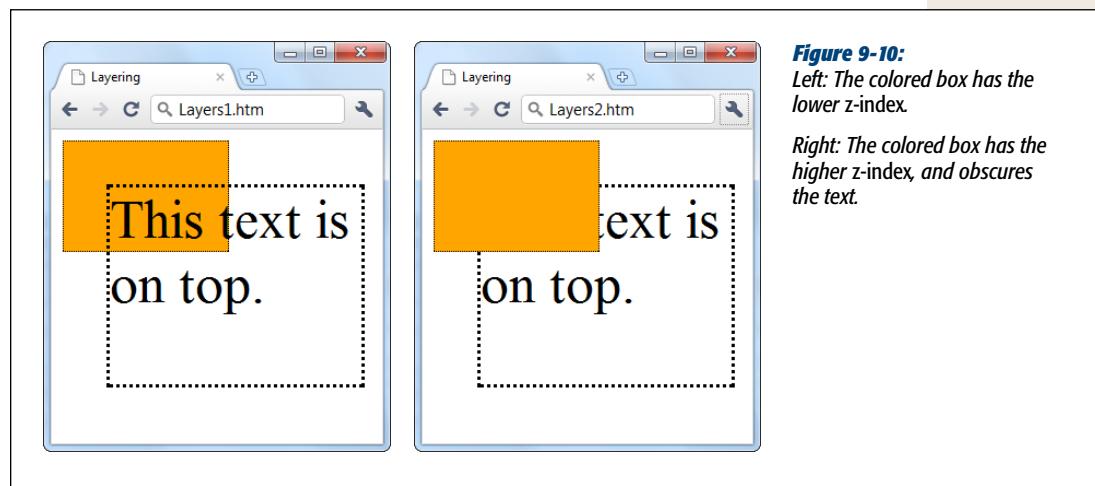


Figure 9-10:

Left: The colored box has the lower *z-index*.

Right: The colored box has the higher *z-index*, and obscures the text.

Combining Absolute and Relative Positioning

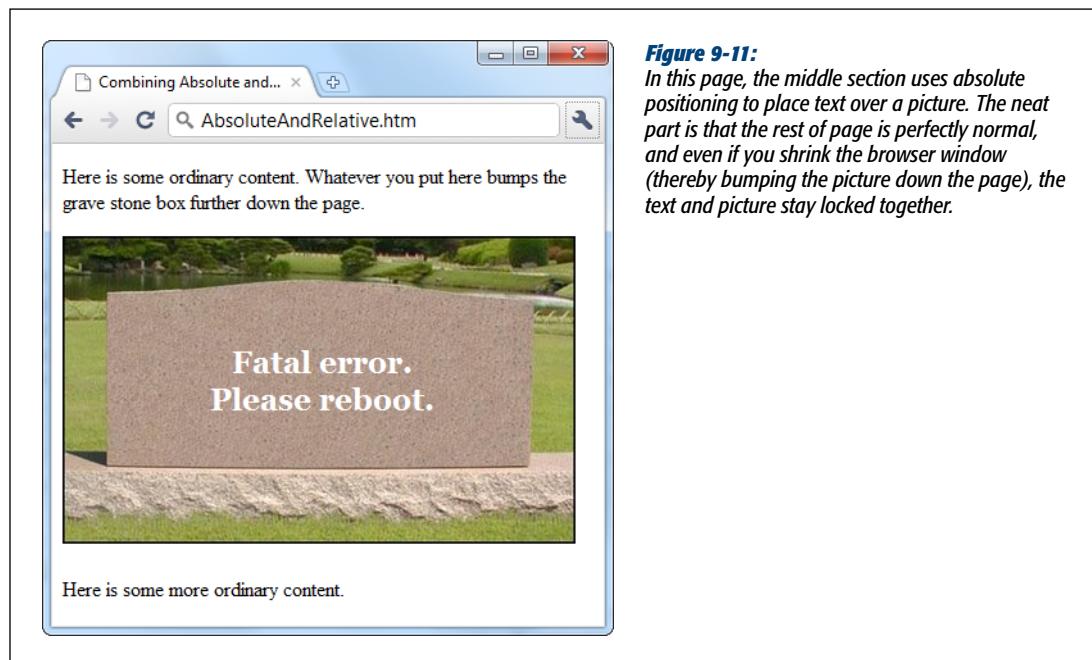
Style sheet experts know that they don't need to stick to just absolute or relative positioning. They can get the best of both worlds with a little careful planning.

To understand how this works, you need to know the following style-sheet secret: When you use absolute positioning, your browser interprets the coordinates *relative to the container*. As you saw in several examples earlier, when you put a `<div>`

element in the <body> section of a page, your browser positions that element in relation to the page. Set the <div> element's left coordinate to 10 pixels, and your browser positions the element 10 pixels from the left edge of the page. But here's a nifty experiment: Try placing the same <div> element inside another element, like a table cell. Now your browser positions the <div> element 10 pixels from the left edge of the table cell, *no matter where you place that table cell on the page*. It's as if the <div> element exists in its own private world—and that's the world of the container it's in, not the world of the main page.

So how can you use this understanding to your advantage? One technique is to use absolute positioning to create a special effect, like text superimposed on a photo. To try this out, create a page with several <div> elements. But don't use absolute positioning—instead, let these <div> elements fit themselves into the page one after the other, the normal web way. In the first and last <div> elements, add ordinary content (text, pictures, and whatever else you like). But in the middle <div> element, let loose with absolute positioning.

Figure 9-11 shows an example. Here, the first <div> element holds an ordinary paragraph, as does the third <div> element. But the middle <div> element uses absolute positioning to add white text over the picture of a tombstone.



Here's the content of the page:

```
<div>
  <p>Here is some ordinary content. Whatever you put here
  bumps the grave stone box further down the page.</p>
</div>

<div class="GraveContainer">
  
  <p class="GraveText">Fatal error.<br />Please reboot.</p>
</div>

<div>
  <p>Here is some more ordinary content.</p>
</div>
```

The middle `<div>` element uses two style rules to apply all the style properties this example needs. The `GraveText` rule turns on the absolute positioning:

```
p.GraveText {
  position: absolute;
  top: 60px;
  left: 115px;
  color: white;
  font-size: x-large;
  font-weight: bold;
  text-align: center;
}
```

The `GraveContainer` rule sizes the `<div>` element. Ordinarily, a `<div>` element enlarges itself to fit its contents. But when you use absolute positioning, the `<div>` element no longer knows how big it should be, and it shrinks itself down to nothing. Here's the rule that gives the `<div>` element the correct height, and ensures that the subsequent content on the page (the third `<div>` element, with the final paragraph) appears in the right place:

```
div.GraveContainer {
  position: relative;
  height: 250px;
}
```

You'll notice the `GraveContainer` uses relative positioning. This allows it to fit into the flow of your web page, without any hassle.

As a general rule, use relative positioning to make sure a page's layout is as flexible and adaptable as possible. But as you see in this example, it's perfectly reasonable to use a `<div>` element to section off smaller regions that use absolute positioning—in fact, doing so gives you the chance to add some nifty effects.

Sizing Tables

The time when tables were used to shape the layout of a page is distant history. However, tables still play a role in web pages as an easy way to show grids of information (like the population statistics in Figure 9-12).

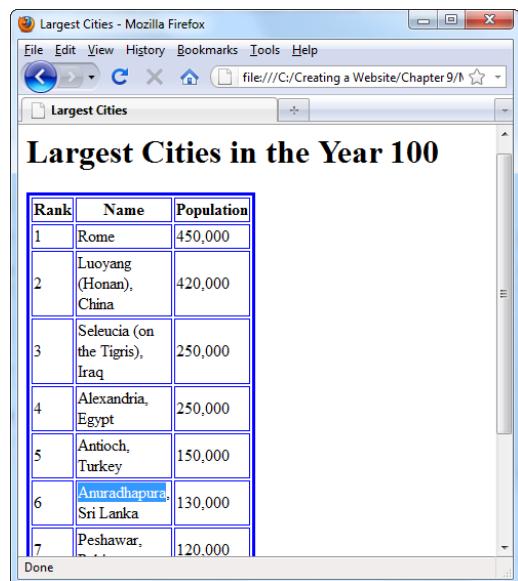


Figure 9-12:

In this example, the style sheet calls for a table width of 1 pixel. But the browser doesn't shrink the table down that far because the content influences the table's minimum size. In this table, the city name Anuradhapura is the longest unsplittable value, so the browser uses that name to determine the width of the column. If you really want to ratchet the size down another notch, try shrinking the text by applying a smaller font size.

Ordinarily, HTML makes a table as wide as necessary to display all its columns, and each column grows just wide enough to fit the longest line of text (or to accommodate other content, like a picture). However, there's one additional rule: The table can't grow wider than the browser window. Once a table reaches the full width of the current window, the browser starts wrapping the text inside each column, so that the table grows taller as you pile in more content.

Of course, there are some circumstances when you need to take control of table sizing. For example, you might want to give more space to one column than another. In this situation, you can rely on the same width and height properties that you used to size `<div>` elements when you organized a page (see page 247).

Sizing a table

In most cases, you want to explicitly set the width of your table and its individual columns. When you do so, the table respects these dimensions and wraps text to accommodate those widths.

When sizing a table, you can use a pixel width or a percentage width. For example, the following rule limits the table to half the width of its current container (which, in an ordinary page, makes it half the width of the page):

```
table.Cities {  
    width: 50%;  
}
```

To display this specific table, you cite the `table` class in your HTML document:

```
<table class="Cities">  
  ...  
</table>
```

The table dynamically resizes as you resize the browser window so it keeps to its half-window width.

If you use exact pixel widths, the table dimensions never change. For example, the following rule creates a table that's a generous 500 pixels wide:

```
table.Cities {  
  width: 500px;  
}
```

There's one important caveat to table sizing: Although you can make a table as large as you want (even if it stretches beyond the borders of a browser window), you don't have the same ability to shrink a table. If you specify a table size that's smaller than the minimum size the table needs to display your content, the table ignores your settings and appears at this minimum size (see Figure 9-12).

Sizing a column

Now that you know how to size a table, you probably want to know what your browser does if a table has more than enough space for its content. Once a table reaches its minimum size (just large enough to fit all its content), your browser distributes any extra space proportionately, so that every column increases in width by the same amount.

Of course, this isn't necessarily what you want. You might want to create a wide descriptive column paired with a narrow column of densely packed text. Or you might want to set columns to a specific size so that all your pages look uniform, even if the content differs.

To set a column's size, you use the `width` property in conjunction with the `<td>` and `<th>` elements. Once again, you can do this proportionately, using a percentage, or exactly, using a pixel width. However, proportional sizing has a slightly different effect when you use it with columns. Earlier, when you used a percentage value for table width, you sized the entire table relative to the width of the page. In that example, you had a table width of 50 percent, which means the table occupied 50 percent of the full width of the page. But when you use a percentage value to set a *column* width, you're defining the percentage of the *table* width that the column should occupy. So when you set a column width of 50%, the column takes up 50 percent of the *table*.

When you size columns, you need to create a style rule for each one, giving it a unique class name (page 147). Each column is potentially a different width; you can't just write a single style rule that applies to every column, unless you want them all to have exactly the same width.

The following style rules set different widths for each column in the table you saw in Figure 9-12.

```
th.Rank {  
    width: 10%;  
}  
th.Name {  
    width: 80%;  
}  
th.Population {  
    width: 10%;  
}
```

In this example, the class names match the column titles, which makes it easy to keep track of which rule applies to which column.

Note: When you use percentage widths for columns, you don't need to specify values for all three columns. If you leave one out, the browser sizes that column to fill the rest of the space in the table. If you do decide to include widths for each column (as in the previous example), make sure that they add up to 100 percent to avoid confusion. Otherwise, the browser will override one of your settings, and you won't know how your table will actually appear.

For these rules to take effect, you need to apply them to the corresponding cells:

```
<table class="Cities">  
    <tr>  
        <th class="Rank">Rank</th>  
        <th class="Name">Name</th>  
        <th class="Population">Population</th>  
    </tr>  
    <tr>  
        <td>1</td>  
        <td>Rome</td>  
        <td>450,000</td>  
    </tr>  
    ...  
</table>
```

Notice that you specify widths only for the column elements in the first row (the ones that contain the cell headers in this example). You could apply the rule to every row, but there's really no point. When the browser builds a table, it scans the whole table structure to determine the required size, based on the cell content and any explicit width settings. If you apply a different width to more than one cell in the same column, the browser simply uses the largest value.

Tip: It's a good idea to size your table by applying style rules to the first row. Doing so makes your HTML more readable, because you can see the dimensions of your table immediately obvious.

Sizing a row

You can size a row just as easily as you size a column. The best approach is to use the *height* property in the <tr> attribute, as shown here:

```
tr.TallRow {  
    height: 100px;  
}
```

When you resize a row, you affect every cell in every column of that row. However, you're free to make each row in the table a different height.

Tip: Need more space inside your table? Style rules make it easy. To add more space between the cell content and its borders, increase the padding property for the <td> and <tr> elements. To add more space between the cell borders and any adjacent cells, up the margin width for the <td> and <tr> elements. Page 153 has more on adjusting these dimensions.

Multipart Pages

As you start building bigger and more elaborate websites, you'll no doubt discover one of the royal pains of website design: making a common ingredient appear on every page.

For example, you might decide to add a menu of links that lets visitors jump from one section of your site to another. You can place these links in a table or a `<div>` element (two techniques shown in Chapter 9) to position them on a page, but either way you face a problem you need to do a fair bit of copying and pasting to display the menu on every page of your site. If you're not careful, one page can end up with a slightly different version of the same menu. When you decide to make a change to the menu, you face the nightmare of updating every one of your pages. Web creators who try this approach don't get out much on the weekend.

There's no simple solution to this problem, but crafty web designers can use a variety of techniques to get around it. In this chapter, you'll learn about the two most accessible solutions: server-side includes and the page template feature in Dreamweaver and Expression Web.

Understanding Multipart Pages

By this point, you've amassed a solid toolkit of tactics and tricks for building web pages. You learned to polish your pages with modern fonts and colors, gussy them up with a trendy layout, and add images and links to the mix. As you apply these techniques to a complete website, however, you'll run into some new challenges.

One of the first hurdles you face when you go from one web page to a dozen or more is how to make them all consistent. Consistent formatting is relatively easy. As long

as you carefully plan the structure of your site and use an external style sheet (see Chapter 6), you can apply a common look and feel to as many pages as you want.

But style sheets have their limits. They can't help you if you need the same *content* in more than one page. That's a problem, because today's websites repeat specific elements on every page, like a header and a set of navigation buttons (see Figure 10-1).

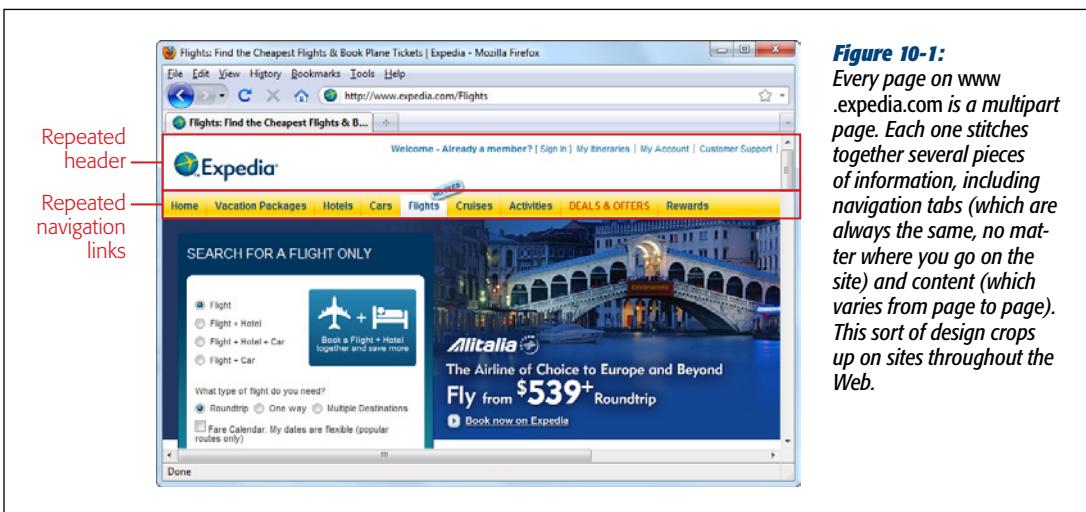


Figure 10-1:
Every page on www.expedia.com is a multipart page. Each one stitches together several pieces of information, including navigation tabs (which are always the same, no matter where you go on the site) and content (which varies from page to page). This sort of design crops up on sites throughout the Web.

So how do web designers create multipart pages? Big websites use content management systems or custom web applications (see the sidebar on page 269). Ordinary web designers need to rely on something simpler. Here are three techniques that solve the problem:

- **Server-side includes.** A server-side include is a command that injects the contents of one HTML file inside another. This lets you carefully separate a block of HTML content (for example, a menu) and reuse it in multiple pages. However, there's a significant caveat—the HTML standard doesn't support server-side includes, so you can use this feature only if you have the right type of web server.
- **Seamless frames.** This new HTML5 feature lets you use an element called <iframe> to inject the contents of one page into another. It's like a server-side include, except the browser makes the magic happen, not the web server. Seamless frames sound like the perfect solution, but hands off for now—at present, no browser fully supports them.
- **Page templates.** Some high-powered web page editors (namely, Dreamweaver and Expression Web) include a page template feature. You begin by creating a template that defines the structure of your web pages and includes the repeating content you want to appear on every page (like a menu or a header). Then you use that template to create all your site pages. Here's the neat part: When you update the template, your web page editor automatically updates all the pages that use it.

Note: In the not-so-distant past, web designers would also use a controversial HTML feature called frames to show multiple-part pages in separate sections of a browser window. However, frames come with a set of notorious problems. For example, they confuse search engines, cause problems with browser history-tracking, and don't adapt well to different screen sizes and mobile devices. The frames feature is now considered obsolete.

HOW'D THEY DO THAT?

Multipart Pages on Big Websites

Popular websites don't seem to have a problem dealing with repeated content. No matter what product you view on Amazon, for example, you see the familiar tabbed search bar at the top. No matter what vacation you check out in Expedia, you keep the same set of navigation tabs. That's because Amazon and Expedia, like almost all of the Web's hugest and most popular sites, are actually *web applications*. When you request a page from one of these sites, a custom-tuned piece of software creates the HTML page on the fly.

For example, when you view a product on Amazon, a web application reads the product information out of a gargantuan database, transforms it into an HTML page, and tops it off with the latest version of the search bar. Your browser displays the end result as a single page. This technique

allows Amazon to assemble any content into a slick web page, without forcing the site designers to maintain thousands (or even millions) of different HTML files. The web application approach is a bit like the server-side include approach described below (in both cases, the process involves assembling the whole page out of pieces on the web server), but it's more elaborate.

Although web applications offer an elegant solution for creating multipart pages, it's a dead end for the casual web-head. To build and maintain a web application of your own, you'd need top-shelf programming skills and a team of IT experts to help you out. So unless you're ready to start a new career as a hard-core code jockey, you're better off using the techniques described in this chapter.

Server-Side Includes

Even though you can't write a web application on your own, you *can* borrow a few tricks from the web applications model—if your web host supports them. The simplest example is a technology called *server-side includes* (SSIs), which is a scaled-down version of the HTML-assembling trick used on sites like Amazon and Expedia.

Essentially, a server-side include is an instruction that tells a web server to insert the contents of one HTML file into another. For example, imagine you want to use the same menu on several pages. You would begin by saving the menu as a separate file, which you could name *menu.htm*. Here are its contents:

```
<h1>Menu</h1>
<a href="...">Page 1</a><br />
<a href="...">Page 2</a><br />
<a href="...">Page 3</a><br />
<a href="...">The End</a>
```

Notice that *menu.htm* isn't a complete HTML document. It lacks elements like *<html>*, *<head>*, and *<body>*. That's because *menu.htm* is a building block that you embed in other, full-fledged HTML pages.

Now you're ready to use the menu in a web page. To do that, you add a specialized *include* command to the page where you want the menu to appear. Here's what it looks like:

```
<!--#include file="menu.htm" -->
```

The Include command disguises itself as an HTML comment (page 39) using the <!-- characters at the beginning of the line and the --> characters at the end. But its core tells the real story. The number sign (#) indicates that this command is actually an instruction for the web server, and the *file* attribute points to the file you want to use.

Here's the Include command at work in a complete web page:

```
<!DOCTYPE html>

<html>
<head>
  <title>Server-Side Include Test</title>
  <link rel="stylesheet" href="styles.css" />
</head>

<body>
  <div class="Header">
    <h1>Templates Rule!</h1>
  </div>

  <div class="MenuPanel">
    <!--#include file="menu.htm" -->
  </div>

  <div class="Content">
    <p>This is the welcome page. Just above this text is the handy menu for this site.</p>
  </div>
</body>
</html>
```

When you request this page, the web server scans through it, looking for instructions. When it finds the Include command, it retrieves the specified file and inserts its contents into that position on the page. It then sends the final, processed file to you. In the current example, that means your web browser receives a web page that actually looks like this (see Figure 10-2):

```
<!DOCTYPE html>

<html>
<head>
  <title>Server-Side Include Test</title>
  <link rel="stylesheet" href="styles.css" />
</head>

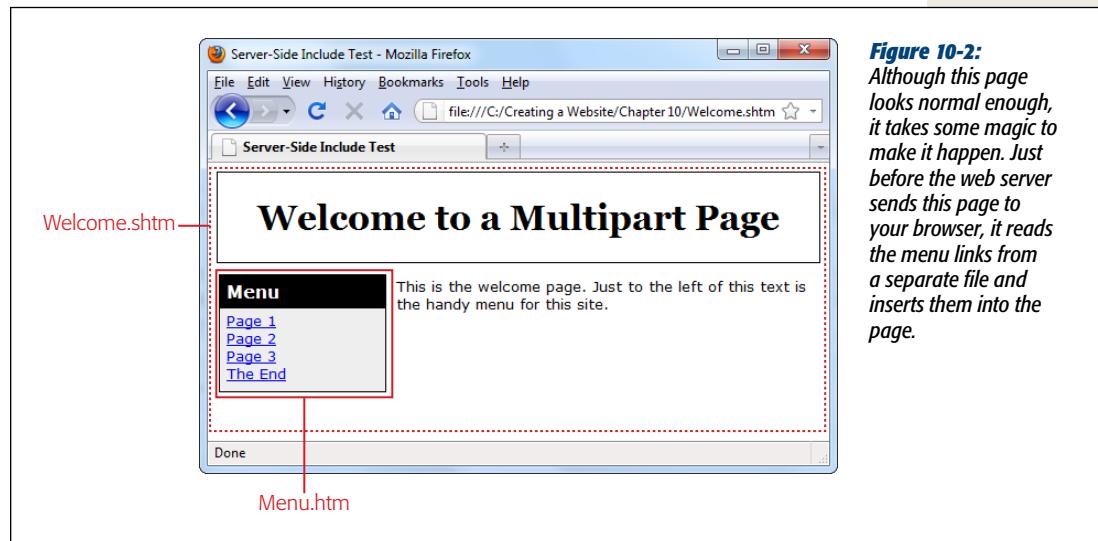
<body>
  <div class="Header">
    <h1>Welcome to a Multipart Page</h1>
  </div>
```

```

<div class="MenuPanel">
  <h1>Menu</h1>
  <a href="...">Page 1</a><br />
  <a href="...">Page 2</a><br />
  <a href="...">Page 3</a><br />
  <a href="...">The End</a>
</div>

<div class="ContentPanel">
  <p>This is the welcome page. Just to the left of this text is the
     handy menu for this site.</p>
</div>
</body>
</html>

```

**Figure 10-2:**

Although this page looks normal enough, it takes some magic to make it happen. Just before the web server sends this page to your browser, it reads the menu links from a separate file and inserts them into the page.

The advantage to this technique is obvious. You can add the `Include` command to as many web pages as you want, and still keep just one copy of your menu. That lets you edit your menu easily, and ensures that all your pages will have the same version of it.

If this discussion sounds a bit too good to be true—well, it is. You may face a number of complications:

- **Web server support.** Not all web servers support server-side includes. To get the lowdown, contact your web hosting company.
- **Page types.** For a server-side include to work, the web server has to process your page and search for `Include` instructions every time a browser requests a page. That happens automatically, but only if you use the right page type. You can't give your pages the `.htm` or `.html` extensions; instead, you need to use `.shtm` or `.shtml` (on an Apache web server), or `.asp` or `.aspx` (on a Microsoft IIS web server). Once again, contact your web hosting company for details.

- **Design difficulties.** Server-side includes only come into effect when there's a web server at work. If you open a web page stored on your hard drive, your browser ignores the Include instruction and you won't see the menu at all. That makes it difficult to test your site without uploading it to a live web server. Dreamweaver gives you partial relief—if you open a web page that uses server-side includes, you'll see the contents of the included files in Dreamweaver's design window while you edit the page.

If you know your web host supports server-side includes and you aren't fazed by the design difficulties, why not give them a whirl? And if they still don't suit you, you have another compromise to consider: page templates (up next).

Note: Because the Include command masquerades as an HTML comment, it won't cause a problem if you put pages with Include instructions on a web server that doesn't support server-side includes. The server simply ignores the SSIs.

Page Templates

So far, you've seen how you can put the same content on a whole batch of pages using server-side includes. This approach is great—if your web hosting company supports server-side includes and you don't mind the design challenges. After all, the alternative (making a separate copy of the repeated content on each page) is a surefire way to fry the last few neurons of your overworked brain.

However, web designers who own one of the two premiere web design tools—Adobe Dreamweaver and Microsoft Expression Web—have one more option. They can create a *page template* that sets out the structure of their site pages, and then reuse that template relentlessly. The technique is similar to server-side includes, but instead of having a web server do the work, you give the task to your web page editing program. That means you don't need to worry about mistakes or web server compatibility. For these reasons, page templates might just be the perfect compromise for small- or medium-sized websites.

Before you get started with templates, it's time to face a few drawbacks:

- **More time.** Every time you change a page template, your web design tool needs to update all the pages that use the template. For this reason, page templates aren't a great idea for huge websites, because the updating process takes too long.
- **More fragile.** As you'll see, the page template system is based on a few secret comments you bury in your HTML pages. Unfortunately, it's all too easy to accidentally delete or move one of these comments and break the link between a page and its template. When using page templates, you need to edit your pages with extra caution.

- **Nonstandard.** Page templates work differently in Dreamweaver and Expression Web. If you use page templates to craft the perfect website in Dreamweaver, you can't switch your site over to Expression Web (or vice versa)—at least not without a painstaking conversion process that you have to carry out by hand.

If you're willing to put up with these shortcomings to create true multipart pages, keep reading.

Understanding Page Templates

The page template systems in Dreamweaver and Expression Web are surprisingly similar. Here are the ground rules:

- You begin by creating a page template. Oddly enough, template files in both programs use the extension .dwt. In Dreamweaver, that stands for Dreamweaver Web Template. In Expression Web, it's short for Dynamic Web Template.
- The template is an ordinary HTML page you use as the basis for every other page on your site. The content you include with the template becomes *fixed content*—your web page editor passes it along to every page you create with the template. If you pop a menu bar into the page template, every page gets that same menu bar. You can modify this fixed content in only the page template itself; you can't edit fixed content in the pages the template creates.
- Along with fixed, unchangeable content, the page template includes *editable regions*—places where you insert each page's unique content. To create one of these regions, you use specialized HTML comments. Although the comments look similar in Dreamweaver and Expression Web, they aren't exactly the same, so you can't use a page template from one program in the other.
- Once you perfect your template, you can create the individual site pages that use it. These pages acquire all the fixed content from the template and supply new content for each editable section of the page. The specialized comments always remain in place. They let your web page editor update the page when the template changes.

To really understand how dynamic web templates work, you need to see them in the context of a complete page. Figure 10-3 shows a suitable candidate—a simple multipart page that shares a header and menu. It closely resembles the server-side include example you saw at the beginning of this chapter (page 269).

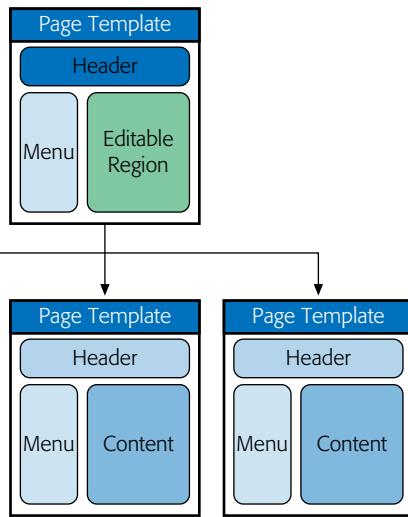


Figure 10-3:

With page templates, your web page editor fuses together a template and the unique content of each page.

In the following sections, you'll build the template for the page shown in Figure 10-4, and create the four pages that use it.

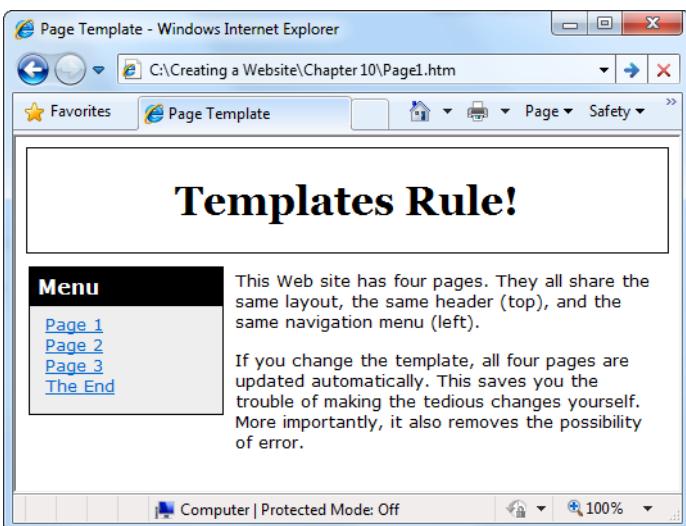


Figure 10-4:

Every page on this website needs the same header and menu.

Creating a New Page Template

Although you can turn any existing page into a page template, it's often easiest to let your web page editor start you out with an example.

Here's how you do that in Dreamweaver:

1. **Before getting started, make sure you define your website, as described on page 97.**

Dreamweaver always puts page templates in the Templates subfolder of your website folder. If you don't define a website, you'll still be able to create a page template, but you won't be able to apply it to other pages, which makes it relatively useless.

2. **Choose File→New.**

Dreamweaver's New Document window appears.

3. **On the left, choose Blank Page. In the Page Type list, choose HTML Template.**

In the Layout list, keep the standard option ("<none>") highlighted. You can use other layouts, but this example assumes you're creating the entire template from scratch.

This is also a good time to pick the doctype you want from the DocType list, so you don't need to change it by hand after you create the template.

4. **Click Create.**

This creates a new page template, with the bare minimum of markup. In the following sections, you'll learn how to customize the template.

After you finish perfecting your template, choose File→Save to open the Save As Template dialog box. From the Site list, pick the defined site where you want to store the template. It starts out with a name like *Untitled_1.dwt*, but you can type in a better name, and then click Save to make it official. Dreamweaver stores the template in the Templates subfolder of your website folder.

Here's how you do the same thing in Expression Web:

1. **Before getting started, make sure you create Expression Web's hidden metadata folders (page 93).**

Expression Web uses these folders to store details about the web pages in your site. For example, it keeps track of all the pages that use a given template, which lets it update these pages when you change the template.

2. **Choose File→New→Page.**

Expression Web opens its New window.

3. **In the list on the left, choose General. In the list on the right, choose Dynamic Web Template.**

At this point, you can click the Page Editor Options link to set additional options, including the automatic doctype.

4. Click OK.

Your page template starts out with a name that's something like *Untitled_1.dwt*. When you save it (by choosing File→Save) you can pick a better name.

You now have a brand-new page template. Currently, it's little more than a basic HTML skeleton. To turn it into something useful, you need to understand a bit more about how page templates work.

The Anatomy of a Page Template

Page templates are completely ordinary HTML pages. The magic happens through specialized comments. Although these look like ordinary HTML comments (page 39), they actually carve the page into separate, editable regions (Figure 10-5).

Figure 10-5:
Editing a template is just like editing an ordinary web page. You can use the same Design view and the same commands. But if you look through the Source view, you'll stumble across the special comments that indicate editable sections.



As you already learned, your template includes fixed content. When you create a new page using this template, you can't change the fixed content. The comments in the template identify the editable sections where you can insert new content. These comments come in pairs, so the first one defines the start of an editable region, while the second one demarcates its end. Here's a comment pair in a Dreamweaver template:

```

<!-- TemplateBeginEditable name="body" -->
...
<!-- TemplateEndEditable -->

```

And here's the same thing in an Expression Web template:

```
<!-- #BeginEditable "body" -->
```

```
...  
<!-- #EndEditable -->
```

There are two things to notice here. First, comments begin with the standard comment indicator <!-- followed by a specific command (like *TemplateBeginEditable* or *#BeginEditable*). That's how your web page editor recognizes that the comment is actually a template instruction. Second, you can see that the comments give your editable region a name. In both these examples, the region is named body.

Note: Because templates use comments, they're a bit fragile. Seemingly minor changes, like deleting one of the comments in a pair, changing a section name, or rearranging comments in the wrong order, can cause problems. At worst, your editor will become so confused that updating the template will erase part of your page. To avoid issues like these, always make a backup of your website before you begin editing it, especially when page templates are involved.

Now that you understand the comment system, you can create a page template for the page shown in Figure 10-4. In this example, the header and navigation bar are fixed, unchangeable elements. The content region is the portion that appears under the header and just to the right of the menu bar.

The most straightforward way to create an editable region is to type the magic comments into Code view on your own. In Dreamweaver, you can also choose Insert→Template Objects→Editable Region. Web designers in a hurry can press the shortcut key combination Ctrl+Alt+V.

Expression Web makes you work a bit harder. First, position your cursor where you want the editable region to go, and then choose Format Dynamic Web→Template→Manage Editable Regions to open the Editable Regions window. Once there, fill in a name for the new editable region and click Add to create it.

Here's the Dreamweaver version of the finished template:

```
<!DOCTYPE html>  
  
<html>  
<head>  
  
    <!-- TemplateBeginEditable name="title" -->  
    <title></title>  
    <!-- TemplateEndEditable -->  
  
    <link rel="stylesheet" href="styles.css" />  
</head>  
<body>  
    <div class="Header">  
        <h1>Templates Rule!</h1>  
    </div>  
  
    <div class="MenuPanel">  
        <h1>Menu</h1>  
        <p>
```

```

<a href="">Page 1</a><br />
<a href="">Page 2</a><br />
<a href="">Page 3</a><br />
<a href="">The End</a>
</p>
</div>

<!-- TemplateBeginEditable name="content" -->
<div class="ContentPanel">
</div>
<!-- TemplateEndEditable -->

</body>
</html>

```

And here's the Expression Web equivalent:

```

<!DOCTYPE html>

<html>
<head>

<!-- #BeginEditable "title" -->
<title></title>
<!-- #EndEditable -->

<link rel="stylesheet" href="styles.css" />
</head>

<body>
<div class="Header">
<h1>Templates Rule!</h1>
</div>

<div class="MenuPanel">
<h1>Menu</h1>

<p>
<a href="">Page 1</a><br />
<a href="">Page 2</a><br />
<a href="">Page 3</a><br />
<a href="">The End</a>
</p>
</div>

<!-- #BeginEditable "content" -->
<div class="ContentPanel">
</div>
<!-- #EndEditable -->

</body>
</html>

```

Notice that this example actually creates *two* editable regions. One is the content that appears to the right of the menu panel, and the other is the title at the top of the browser window. Thanks to this latter detail, you can give all your pages a unique title.

You'll also notice that both editable regions include some content (like the tags for the `<title>` element or a `<div>` element). When you create a page from this template, the editable regions always includes these elements. However, you're free to change or replace them with something completely different.

Using a Page Template

Once you finish your template, you're ready to put it into action. The process is similar in both programs. Here's how it goes down in Dreamweaver:

1. Choose File→New.

Dreamweaver opens a New Document window.

2. On the left of the New Document window, choose Page From Template. Then, in the Site list, choose your website.

The program lists the templates in your website's Templates folder.

3. Select the template you want, and then click Create.

You see the markup for both fixed and editable content in your new page. As noted above, you won't be able to change the fixed content; Dreamweaver displays it in light gray to remind you.

Make sure you keep the “Update page when template changes” checkbox selected. This way, when you change your template, Dreamweaver updates all the pages that use it.

Here's the same task in Expression Web:

1. Choose File→New.

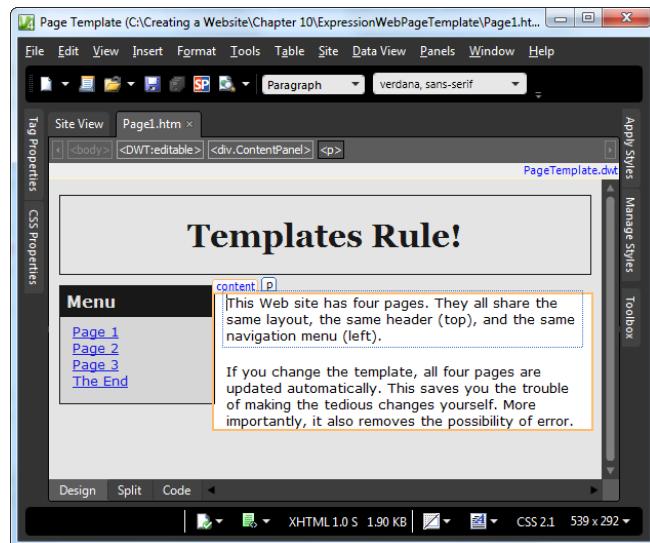
Expression Web opens its New window.

2. In the first list, choose General. In the second one, choose “Create from Dynamic Web Template.” Then click OK.

The program opens Attach Dynamic Web Template window.

3. Browse to the .dwt file you created in the previous section, select it, and then click Open.

Expression Web displays the markup for both fixed and editable content in your new page. It highlights fixed content with a yellow background. (Both Dreamweaver and Expression Web make fixed content even more distinct in Design view, as shown in Figure 10-6.)

**Figure 10-6:**

In Design view, both programs identify editable regions by drawing a box around them and displaying the name of the region in a tiny label at the top of the page ("content" in this example). You can click an editable region to type in content, but you can't click anywhere else on the page.

To create the page shown in Figure 10-4, you simply add a title and a couple of paragraphs of text in the editable content region. Here's the finished page in Dreamweaver, with the new additions highlighted:

```
<!DOCTYPE html>

<html>
<!-- InstanceBegin template="/Templates/PageTemplate.dwt"
codeOutsideHTMLIsLocked="false" --&gt;
&lt;head&gt;
    &lt;!-- InstanceBeginEditable name="title" --&gt;
    &lt;title&gt;Page Templates&lt;/title&gt;
    &lt;!-- InstanceEndEditable --&gt;

    &lt;link rel="stylesheet" href="styles.css" /&gt;
&lt;/head&gt;

&lt;body&gt;
    &lt;div class="Header"&gt;
        &lt;h1&gt;Templates Rule!&lt;/h1&gt;
    &lt;/div&gt;

    &lt;div class="MenuPanel"&gt;
        &lt;h1&gt;Menu&lt;/h1&gt;
        &lt;p&gt;
            &lt;a href=""&gt;Page 1&lt;/a&gt;&lt;br /&gt;
            &lt;a href=""&gt;Page 2&lt;/a&gt;&lt;br /&gt;
            &lt;a href=""&gt;Page 3&lt;/a&gt;&lt;br /&gt;
            &lt;a href=""&gt;The End&lt;/a&gt;
        &lt;/p&gt;
    &lt;/div&gt;
</pre>

```

```
<!-- InstanceBeginEditable name="content" -->
<div class="ContentPanel">
    <p> This website has four pages. They all share the same layout,
        the same header (top), and the same navigation menu (left).</p>
    ...
</div>
<!-- InstanceEndEditable -->

</body>

<!-- InstanceEnd --></html>
```

When you use a template to build individual pages in Dreamweaver, the new page's comments are slightly different from those in the original page. For example, Dreamweaver replaces the TemplateBeginEditable instruction with an InstanceBeginEditable command.

In Expression Web, the new page gets exactly the same comments as the original template, as shown here:

```
<!DOCTYPE html>

<html>

<!-- #BeginTemplate "PageTemplate.dwt" -->
<head>
    <!-- #BeginEditable "title" -->
    <title>Page Templates</title>
    <!-- #EndEditable -->

    <link rel="stylesheet" href="styles.css" />
</head>

<body>
    <div class="Header">
        <h1>Templates Rule!</h1>
    </div>

    <div class="MenuPanel">
        <h1>Menu</h1>
        <p>
            <a href="">Page 1</a><br />
            <a href="">Page 2</a><br />
            <a href="">Page 3</a><br />
            <a href="">The End</a>
        </p>
    </div>

    <!-- #BeginEditable "content" -->
    <div class="ContentPanel">
        <p> This website has four pages. They all share the same layout,
            the same header (top), and the same navigation menu (left).</p>
        ...
    </div>
    <!-- #EndEditable -->
```

```
</body>
<!-- #EndTemplate -->
</html>
```

Note: When you're ready to upload your website to a server, remember that you want to upload your web *pages*, not your templates. The templates are for your design convenience, and you use them only on your computer.

This example shows just a single page. You see the real advantages of a template when you create dozens of pages based on it. In every case, to create a new page, you need do nothing more than set a title and add a bit of content.

But you'll see the biggest benefit when you change the original template. For example, imagine you modify the template to use a spiffy new graphic for its header:

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

Once you save your changes, your web page editor asks if you want to apply those changes to all the linked pages in the current website. Say Yes and the editor quickly and quietly opens all the pages that use the template and updates them with the new content. The result is an instant facelift for your site (see Figure 10-7).

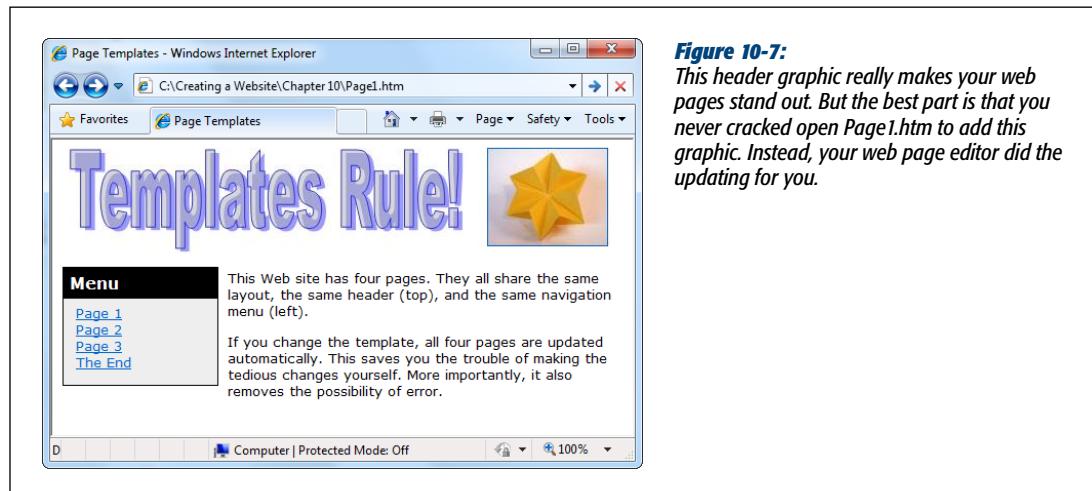


Figure 10-7:
This header graphic really makes your web pages stand out. But the best part is that you never cracked open Page1.htm to add this graphic. Instead, your web page editor did the updating for you.

Note: Although this chapter gives you a solid overview of the page template system, you may need to consider other subtleties. For example, you may make changes to a page template so dramatic that the edited template becomes incompatible with the pages that currently use it. Or, you might want to rewire an existing page to use a different template. To learn about the finer points of page templates, consult a dedicated resource for your web editor of choice. Dreamweaver fans can pick up a copy of *Dreamweaver CS5: The Missing Manual*. Expression Web-ers can check out a free e-book dedicated to the topic of templates at <http://tinyurl.com/9whbc7>.

Introducing Your Site to the World

So far, you've polished your website-design mettle and learned how to build sleek and sophisticated pages. Now it's time to shift to an entirely different role, that of a website *promoter*.

One of the best ways to attract new visitors to your site is to turn up as a result in a web search. For that to happen, the leading search engines (that's Google, Yahoo, and Bing) need to know about your site, and they need to think it's important enough to rank as a search result. For example, if you're hawking fried delicacies at www.sweetsaltsweets.com, you want your website to turn up when someone searches for "chocolate-covered potato chips." And ideally, that result will be among the first few listed, or at least on the first page of results, which greatly increases your odds of getting noticed.

In this chapter, you'll see how search engines work. You'll learn how to make sure they regularly index your site, capture the right information, and expose your brilliance to the world. (Later, in Chapter 12, you'll continue this journey and learn more promotion strategies, all of which can help you work your way up the search results rankings.)

Lastly, you'll learn to gauge the success of your site with *visitor tracking*. You'll use a powerful, free service called Google Analytics to learn some of your visitors' deepest secrets—like where they live, what browsers they use, and which of your web pages they find absolutely unbearable. With this information, you'll have all the tools you need to improve your content and fine-tune the organization of your website. Before you know it, you'll be more popular than chocolate ice cream.

Your Website Promotion Plan

Before you plunge into the world of website promotion, you need a plan. Grab a pencil and plenty of paper, and get ready to jot down your ideas for global website domination (fiendish cackling is optional).

Although all webmasters have their own tactics, web mavens generally agree that the best way to market a site is to follow these steps:

- 1. Build a truly great site.**

If you start promoting your site before there's anything to see, you're wasting your effort (and probably burning a few bridges). Nothing says "never come back" like a website that consists of an "Under Construction" message.

- 2. See step 1.**

If in doubt, keep polishing and perfecting your site. Fancy graphics aren't the key concern here. The most important detail is whether you have some genuinely useful content. Ask yourself—if you were browsing the Web, would you stop to take a look at this site? Make sure you take the time to add the kinds of features that keep visitors coming back. One great option: include a discussion forum (see the next chapter for details on that).

- 3. Make your site search-engine friendly.**

There are a number of ways to tweak and optimize your site to help search engines understand the nature of your content. Small details like page titles, alternate text, and meta elements are easy to overlook when you start building a site, but become more important when you need to popularize it.

- 4. Submit your website to Internet directories.**

Like search engines, directories help visitors find websites. The difference between directories and search engines is that directories are generally smaller catalogs put together by humans, rather than huge sprawling text indexes amassed by computers. Search engines may give preferential treatment to websites that turn up in directory listings.

- 5. Submit your site to Internet search engines.**

Now you're ready for the big time. Once you submit your site to search heavyweights like Google and Yahoo, it officially enters the public eye. However, it takes time to climb up the rankings and get spotted.

- 6. Tweak your website's public profile with the Google Webmaster Tools.**

These handy tools let you adjust how Google sees your site and reveals some valuable information about it. For example, it lets you specify the geographic region you want associated with your site, and it lets you discover whether Google's having trouble indexing some of your pages.

7. Figure out what happened.

To assess the successes and failures of your promotion strategies, you need to measure some vital statistics—how many people visit your site, how long they stay, and how many visitors come back for more. To take stock, you need to crack open tools like hit counters and server logs.

You'll tackle these steps throughout this chapter.

Making Your Site Search-Engine Friendly

A “search-engine friendly” website is one that search engines understand. As explained in the box below, a search engine needs to be able to pull some essential information from your web pages. Then it analyzes those details and decides how useful your site is to millions of web searchers every minute.

Modern search engines handle this task quickly and quietly, without revealing how much effort it really is. But behind the scenes, search engines use a lot of messy logic and number-crunching to analyze raw website data. If you can simplify the engines’ work and make the content, quality, and relevance of your site stand out, you just might be rewarded with higher placement in search results.

UP TO SPEED

How Web Search Engines Work

A web search engine like Google has three pieces. The first is an automated program that roams the Web, downloading everything it finds. This program (often known by more picturesque names like *spider*, *robot*, *bot*, or *crawler*) eventually stumbles across your site and copies its contents.

The second piece is an indexer that chews through web pages and extracts a bunch of meaningful information, including the page’s title, description, and keywords. The indexer also records a great deal more esoteric data. For example, a search engine like Google keeps track of the words that crop up most often on a page, what other sites link to your page, and so on. The indexer inserts all this digested information into a giant catalog (technically, a *database*).

The search engine’s final task is the part you’re probably most familiar with—the front-end, or search home page. You enter the keywords you’re hunting for, and the search engine scans its catalog looking for suitable pages. Different engines have different ways of choosing pages, but the basic idea is to make sure the best and most relevant pages turn up early in the search results. (The *best* pages are those that the search engine ranks as highly popular and well-linked. The *most relevant* pages are those that most closely match the search keywords.) Due to the complex algorithms search engines use, a slightly different search (say, “green tea health” instead of just “green tea”) can get you a completely different set of results.

Choose Meaningful Page Titles

Remember that short snippet of content that goes in the <title> element in your web page? It’s easy to overlook, but vitally important, because the title text becomes the key piece of identifying information people see in a search result page (Figure 11-1).

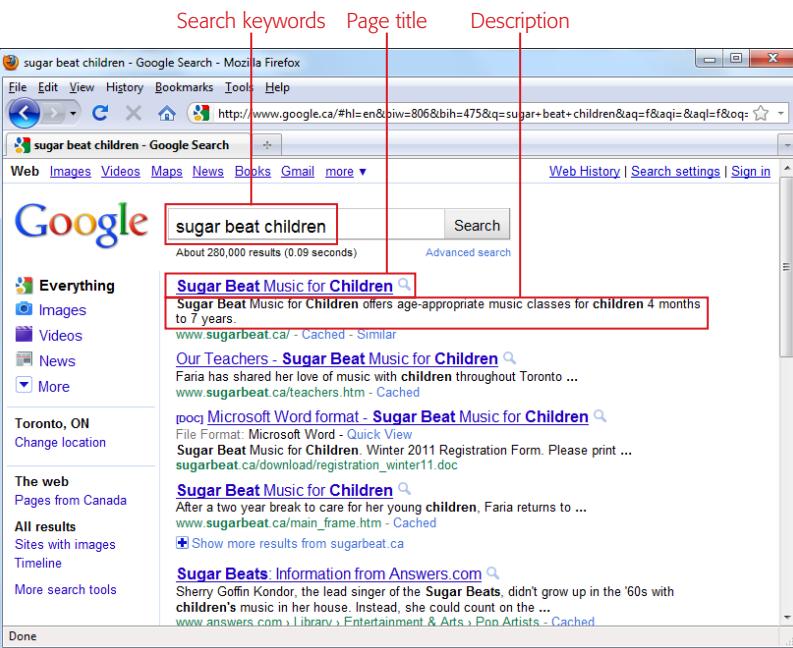


Figure 11-1:
Ever wondered where the information you see in search listings comes from? The underlined link in this example ("Sugar Beat Music for Children") is the title of the web page the search engine found. It pulled the site's description (shown underneath the title) directly from one of the page's hidden meta elements (page 287).

A common newbie mistake is to give every page the same title—for example, a company name. This is appropriate on your home page, but not anywhere else on your site. Instead, a good page title describes the function of your page. For example, a music school named Cacophony Studios might have page titles like this: Our Teachers, The Cacophony Studios Difference, Signing Up for Music Class, Payments and Policies, and so on. (If you really must have the company name in a page title, place it after the descriptive text, as in “Our Teachers - Cacophony Studios.”)

Tip: If a visitor bookmarks your page, the text in the bookmarks menu comes from the <title> element in your page. Keep that in mind, and refrain from adding long slogans. “Ketchup Crusaders – Because ketchup isn’t just for making food tasty” is about the longest you can stretch a title, and even that’s iffy. On the other hand, your titles should include essential information; titles like Welcome or Untitled 1 (a favorite in Expression Web) aren’t very helpful.

Title text is also important because search engines search this text. In fact, they give it more weight than the text inside your page. So if someone performs a search that has keywords that match your title, odds are your page will get better placement in the search results. (But don’t try to game the system with ridiculously long page titles, because search engines may start to compensate by ignoring them.)

Include a Page Description

Along with the title, every web page should have one other basic piece of information: a description that briefly summarizes the page. Once again, the description text plays two roles. First, search engines give it extra weight in a search. Second, they often display the description on the search results page, as shown in Figure 11-2.

The figure shows two screenshots of Google search results. The top screenshot is for the query "sugar beat piano". It displays three search results. The first result is for "Sugar Beat Music for Children - Piano Lessons", showing a snippet about learning piano and a link to the website. The second result is for "Sugar Beat Music for Children - Piano Orff Experience", showing a snippet about the Orff Experience curriculum and a link. The third result is for "Sugar Beat Music for Children - Piano Orff Experience", showing a snippet about a new baby and a link. The bottom screenshot is for the query "sugar beat music". It displays one search result for "Sugar Beat Music for Children", showing a snippet about age-appropriate music classes for children and a link.

Figure 11-2:
If a search engine finds matching keywords in a snippet of your page, it displays that snippet, as shown here (top). But if you type in part of a domain name, or if the search engine doesn't have any content that seems relevant, it shows the page's description instead (bottom).

To add a description, you need to supply something called a *meta element* (also known as a *meta tag*). Technically, meta elements contain hidden information—information that doesn’t appear in a visitor’s browser, but may convey important information about your site’s content to something else (like an automated search-bot).

Note: Fun fact for etymologists and geeks alike: the term “meta element” means “elements *about*,” as in “elements that provide information *about* your web page.”

You put all meta elements in the <head> section of a page. Here’s a sample meta element that assigns a description to a web page:

```
<!DOCTYPE html>

<html>
<head>
<meta name="description"
      content="Noodletastic offers custom noodle dishes made to order." />
<title>Noodletastic</title>
</head>
```

```
<body>...</body>  
</html>
```

All meta elements look more or less the same. The element name is `<meta>`, the *name* attribute indicates the type of meta element it is, and the *content* attribute supplies the relevant information. In theory, there's no limit to the types of information you can put inside a meta element. For example, some web page editing programs insert meta elements that say its software built your pages (don't worry; once you understand meta elements, you'll recognize this harmless fingerprint and you can easily remove it). Another page might use a meta element to record the name of the web designers who created it, or the last time you updated the page.

Tip: Although you can stuff a lot of information into your description, it's a good idea to limit it to a couple of focused sentences that total no more than around 50 words. Even if your description appears on a search results page, readers see only the first part of it, followed by an ellipsis (...) where it gets cut off.

NOSTALGIA CORNER

The Keyword Meta Element

Long ago, when life was far simpler and Nigerian gentlemen never needed help transferring large sums of money, search engines had an easy task. Many ignored most of a web page, but paid close attention to the *keywords* meta element. Then, when someone went searching, the search engine simply checked for pages that had the search terms in their keyword list and put those in the search results.

Here's an example of a typical keywords list. It includes about 25 words or phrases that best represent the website, with each entry separated by a comma.

```
<meta name="keywords"  
content="Noodletastic, noodles, noodle, pasta,
```

delicious, Italian food, fantastic noodles,
ramen, custom, made-to-order, dishes, organic,
whole-wheat, spelt" />

Today, most search engines ignore the keyword list. That's because it was notorious for abuses (many a webmaster stuffed his keyword list full of hundreds of words, some only tangentially related to the site's content). Search engines like Google take a more direct approach—they look at all the words on your web page, and pay special attention to words that appear more often, appear in headings, and so on. Most web experts argue that the keyword list has outlived its usefulness, and many don't bother adding it to their pages at all.

Supply Alternate Text for All Your Images

A search engine draws information from many parts of your page. One easily overlooked detail is alternate text—the text a browser displays if it can't retrieve an image. As you learned on page 184, you specify this text using the *alt* attribute in the `` element:

```

```

Search engines pay attention to alternate text. If a web search uses keywords that match your alternate text, there's a greater chance of your page turning up in the results. The odds increase even more if the searcher is looking for a picture. For example, Google uses alternate text as the basis for its image search tool at <http://images.google.com>. If you don't have alternate text, Google has to guess what the picture is about by looking at nearby text, which is less reliable.

Use Descriptive Link Text

The famous anchor tag can wrap any piece of text. Search engines like Google give that piece of text—the clickable bit that's underlined in blue—extra weight. But that's not much help if you waste it with a link like this:

To learn more about elephants, [click here](ElephantStories.htm).

But a better-designed link like this tells search engines more about your site:

To learn more about elephants, visit our
[Elephant Stories](ElephantStories.htm) page.

It's also worth your while to make sure that the web pages you link to (in this case, that's *ElephantStories.htm*), have good, readable names.

Don't Try to Cheat

There are quite a few unwholesome tricks that crafty web weavers have used to game the search engine system (or at least try). For example, they might add a huge number of keywords, but hide the text so that it isn't visible on the page (white text on a white background is one oddball option, but there are other style-sheet tricks, too). Another technique is to create pages that aren't really part of your website, but that you store on your server. You can fill these pages with repeating keywords. To implement this trick, you use a little JavaScript code to make sure real people who accidentally arrive at the page are directed to the entry point of your site, while search engines get to feast on the keywords.

As seductive as some of these tricks may seem to lonely websites (and their owners), the best advice is to avoid them altogether. The first problem is that they pose a new set of headaches and technical challenges, which can waste hours of your day. But more significantly, search engines learn about these tricks almost as fast as web developers invent them. If a search engine catches you using them, it penalizes your site (so it ranks lower in the search results) or bans it completely, relegating it to the dustbin of the Web. (It even happens to the heavyweights. For example, J. C. Penney was given a hefty search demotion for using a few dirty search-engine optimization tricks during the Christmas 2010 season.)

If you're still tempted, keep this in mind: Many of these tricks just don't work. In the early days of the Web, primitive search engines gave a site more weight based on the number of times a keyword cropped up, but modern search engines like Google use much more sophisticated page-ranking systems. A huge load of hidden keywords won't move you up the search list one iota.

Registering with Directories

Directories are searchable site listings with a difference: humans, not programs, create them. That means a small army of workers painstakingly puts together a collection of sites, neatly sorted into categories. The advantage of directories is that they're well-organized. A couple of clicks can get you a complete list of California's regional newspapers, for example. The unquestioned disadvantage is that directories are dramatically smaller than *full-text* search catalogs. That means directories aren't very useful for those in search of a piece of elusive information that doesn't easily fall into a category, like a list of the English language's most commonly misspelled words. Over the years, as the Web has ballooned in size, directories have become increasingly marginalized, and full-text search tools like Google and Yahoo have become the most common way that people hunt for information.

So, given that directories are just the unattractive cousins of full-text search engines, why do you need to worry about them? Two reasons. First, some web visitors still use directories, even if they don't use them as often as they do full-text search engines. And second, some search engines (including Google) pay attention to directory listings, and tend to rank sites higher if they turn up in certain directories. Getting into the right directories can help you start to move up the results list in a full-text search. And just like college, getting into a directory requires that you submit an application, which you'll learn about next.

The Open Directory Project

The most important directory to submit your site to is the Open Directory Project (ODP) at <http://dmoz.org>. The ODP is a huge, long-standing website directory staffed entirely by thousands of volunteer editors who review submissions in countless categories. The ODP isn't the most popular web directory (that honor currently goes to Yahoo), but other search engines use it behind the scenes. In fact, Google bases its own directory service (<http://directory.google.com>) on the ODP.

Note: In recent years, the ODP has become swamped with registration requests. In some categories, it may be months or years before a site is considered. And there's been more than one sordid story of ODP editors demanding bribes to get a website into the directory. The best advice is this: When your website is ready, follow the steps in this section to submit it to the ODP. However, don't panic if your site doesn't get listed, as search engines still have plenty of other ways to find and rank your pages.

Before submitting to the ODP, take the time to make sure you do it right. An incorrect submission could result in your website not getting listed at all. You can find a

complete description of the rules at <http://dmoz.org/add.html>, but here are the key requirements:

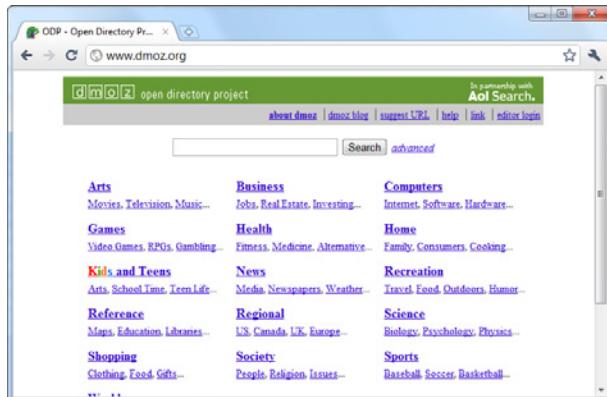
- Don't submit your site more than once.
- Don't submit your site to more than one category.
- Don't submit more than one page or section of your site (unless you have a really good reason, like the separate sections are notably different).
- Don't submit sites that contain "illegal" content. By the OPD's definition, this is more accurately described as unsavory content like pornography, libelous content, or material that advocates illegal activity—you know who you are.
- Clean up any broken links, outdated information, or any other red flags that might suggest to an editor that your site isn't here for the long term.
- When you submit your site, describe it carefully and accurately. Don't promote it. In other words, "Ketchup Masters is a manufacturer of gourmet ketchup" is acceptable. "Ketchup Masters is the best food-oriented site on the Web—the Louisville Times says you can't miss it!" isn't.
- Don't submit an incomplete site. Your "Under Construction" page won't get listed.

The next step is to spend some time at the ODP site, until you find the single best category for your website (see Figure 11-3).

Once you find the perfect category, click the "suggest URL" link at the top of the page and fill out the submission form. It asks for your URL, the title of your site, a brief description, and your email address.

Tip: If you have some free time on your hands, you can offer to help edit a site category; just click the "become an editor" link. And even if you don't have editorial aspirations, why not check out the editor guidelines at <http://dmoz.org/guidelines/> to get a better idea of what's going on in the mind of ODP editors, and how they evaluate your submission?

Once you submit your site, there's nothing to do but wait (and submit your site to the other directories and search engines discussed in this chapter). If a month passes without your site appearing in the listing and you haven't received an email describing any problems with it, it's time to contact the category editor. If that still doesn't work, it's time to contact the category editor. Write a polite email asking why your site wasn't added to the listings, and include the date of your submission(s) and the name, URL, and description of your site. You can find the email address for the category editor at the very bottom of the category page (see Figure 11-4).



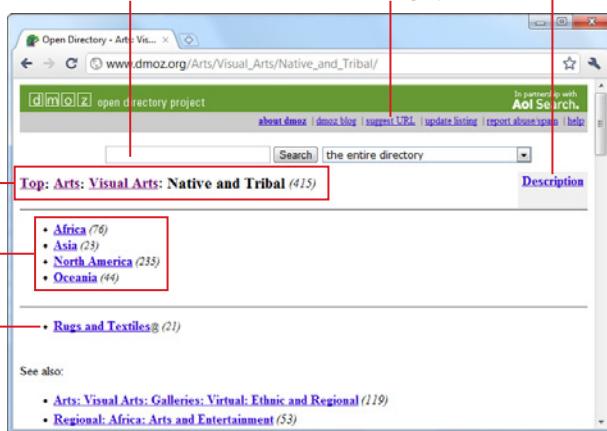
Subcategories in this category.
The number in brackets is the
number of sites they contain.

Get an overview of this
category (Native and Tribal)
to see if you fit in

The hierarchy for the current category

Search for a
site or topic

Add your site to
this category



A related category elsewhere in the directory

Figure 11-3:

Top: When you arrive at the ODP site, you see a group of general, top-level categories.

Bottom: As you click your way deeper into the topic hierarchies, you'll eventually find a specific subcategory that would make a good home for your site, such as the Arts→Visual Arts→Native and Tribal category. There are several subcategories (like Asia, with 22 sites). Categories with an @ after their names link to related categories in a different place in the directory.

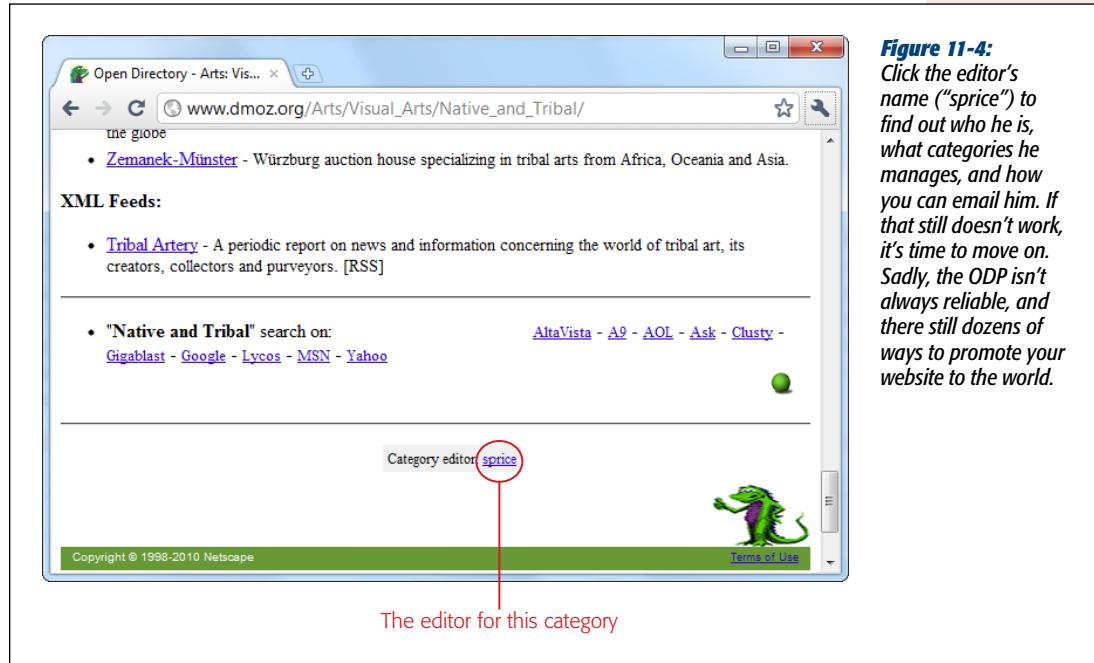


Figure 11-4:
Click the editor's name ("sprice") to find out who he is, what categories he manages, and how you can email him. If that still doesn't work, it's time to move on. Sadly, the ODP isn't always reliable, and there still dozens of ways to promote your website to the world.

The Yahoo Directory

ODP is a great starting point, but it isn't the only directory on the block. The other heavyweight is the Yahoo directory (<http://dir.yahoo.com>). Unfortunately, getting your site into the Yahoo directory takes considerably more work.

First, there's the issue of cost. If you create a noncommercial site, you can probably get in free, but it may take persistence, emails, multiple submissions, and a bit of luck. If you create a commercial site (one whose primary purpose is to make money) and you want to register it with the U.S. Yahoo directory, you need to pay an annual fee of several hundred dollars. And in the ultimate case of adding insult to injury, you won't get your money back if Yahoo rejects your site.

To get started, you can review Yahoo's official submission guidelines at <http://tinyurl.com/47quxhk>. However, you'll be much happier with the *unofficial* write-up at www.apromotionguide.com/yahoo.html, which discusses your free and for-fee options, and explains what Yahoo's cryptic rejection emails really mean. And if you have a commercial website, or you just don't want to suffer through the slow and unreliable free registration process, you need to use the Yahoo Directory Submit service (formerly called Yahoo Express), which is described at <https://ecom.yahoo.com/dir/submit/intro>.

Once you're done with directories (or just ready to move on), it's time to take a look at full-text search engines.

Tip: Web experts are divided about whether a paid Yahoo directory listing is worth the fee. The best advice is this: A Yahoo directory listing can improve your web search standing and bring a respectable increase in traffic, but only if you've created a top-quality website. If you attempt to submit a mediocre site, you're likely to get denied and lose your submission fee altogether.

Registering with Search Engines

For most people, search engines are the one and only tool for finding information on the Web. If you want the average person to find your site, you need to make sure it's in the most popular search engine catalogs and turns up as one of the results in relevant searches. This task is harder than it seems, because the Web is full of millions of sites jockeying for position. To get noticed, you need to spend time developing your site and enhancing its visibility. You also need to understand how search engines rank pages (see the box on page 295 for more information).

The undisputed king of web search engines is Google (www.google.com). It's far and away the Web's most popular search tool, with a commanding share of over 70 percent of all search traffic.

Second place currently goes to Yahoo, and third place to Microsoft's Bing. But behind the scenes, Yahoo quietly uses Bing to power its search results. This means that just two search engines (Google and Bing) are responsible for at least 95 percent of the web searches in the Western world. Other countries, particularly those with web censorship practices, like China, have their own search engines.

It's not too difficult to get Google and other search engines to notice your site. By the time your site's about a month old, Google will probably have stumbled across it at least once, usually by following a link from another site or from the ODP. As described in the box on page 295, Google takes outside links into consideration when sizing up a site, so the more sites that link to you, the more likely you are to turn up in someone's search results.

If you're impatient or you think Google's passing you by, you can introduce yourself directly using the submission form at www.google.com/addurl (see Figure 11-5). To get Bing's attention, go to www.bing.com/webmaster/SubmitSitePage.aspx. Most search engines include a submission form like this. Just make sure you keep track of where you apply, so you don't inadvertently submit your site to the same search engine more than once.

The screenshot shows a Windows Internet Explorer window with the address bar at <http://www.google.com/addurl/?continue=/addurl>. The main content area is titled "Add your URL to Google" and features a "Share your place on the net with us." section. It instructs users to add and update new sites to the index each time they crawl the web. A note says, "Please enter your full URL, including the http:// prefix. For example: <http://www.google.com/>. You may also add comments or keywords that describe the content of your page. These are used only for our information and do not affect how your page is indexed or used by Google." Below this, a "Please note:" section states that only the top-level page from a host is necessary; Googlebot will find the rest. It also mentions that updates to its index are regular, so outdated link submissions are not necessary. A "Comments:" field contains "http://www.SamMenzesHomemadePasta.com". To the right, there are "Other Options" like "Instant Ads on Google", "Google AdSense for Web Publishers", and "Google-Quality Site Search". The status bar at the bottom shows "Internet | Protected Mode: On".

Figure 11-5:
You can safely skip
the comments sec-
tion on this page,
but make sure you
include the <http://>
prefix at the start of
your web page's URL.

UP TO SPEED

How Google's PageRank Works

Google uses a rating system called *PageRank* to size up different web pages. It doesn't use PageRank to *find* search results, it uses it to *order* them. When you execute a search with Google, it pulls out all the sites that match your search keywords. Then it orders the results according to the PageRank of each page.

The basic idea behind the PageRank system is that it determines the value of your site by the community of other websites that link to it. There are a few golden rules:

- The more sites that link to you, the better.
- A link from a more popular site (a site with a high PageRank) is more valuable than a link from a less popular site.

- The more links a site has, the less each link is worth. In other words, if someone links to your site and just a handful of other sites, that link is valuable. If someone links to your site and *hundreds* of other sites, the link's value is diluted.

Although Google regularly fine-tunes its secret PageRank recipe, web experts spend hours trying to deconstruct it. For some fascinating reading, you can learn more about how PageRank works (loosely) at www.markhorrell.com/seo/pagerank.html. For a more technical look at the math behind the PageRank algorithm, check out <http://en.wikipedia.org/wiki/PageRank>.

Rising Up in the Search Rankings

You'll soon discover that it's not difficult to get into Google's index, but you might find it exceedingly hard to get noticed. For example, suppose you submit the site www.SamMenzesHomemadePasta.com. To see if you're in Google, try an extremely specific search that targets just your site, like "Sam Menzes Homemade Pasta." This should definitely lead to your doorstep. Now, try searching for just "Homemade Pasta." Odds are, you won't turn up in the top 10, or even the top 100.

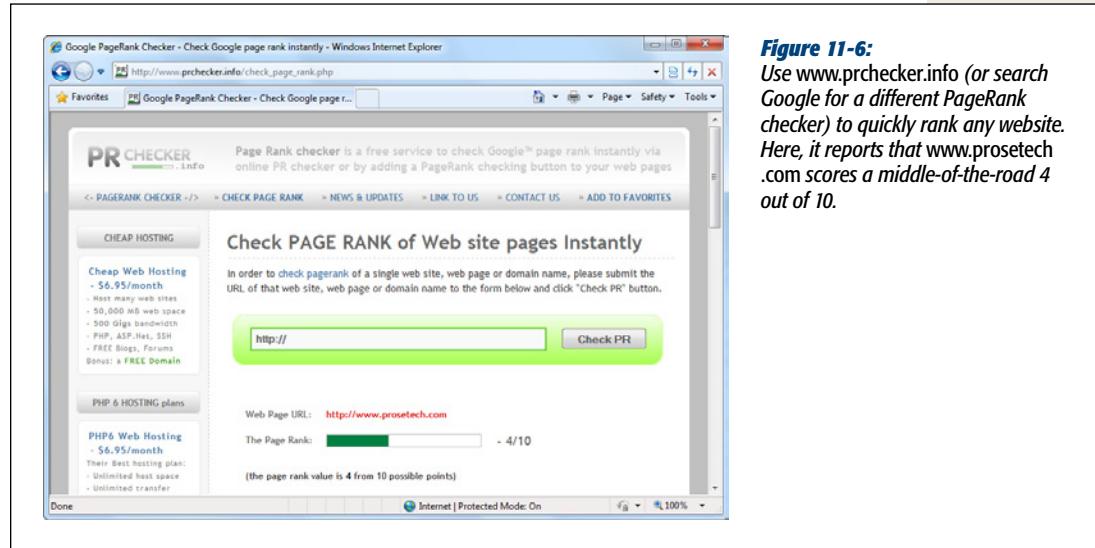
So how do you create a site that the casual searcher's likely to find? There's no easy answer. Just remember that the secret to getting a good search ranking is having a good PageRank, and getting a good PageRank is all about connections. To stand out, your website needs to share links with other leading sites in your category. It also helps to have people talking about your site online, whether that's on Twitter, Facebook, or a personal blog. In the next chapter, you'll get plenty of tips for website promotion.

Google won't tell you the PageRank of your website. (In fact, they suggest you concentrate on building a fantastic website rather than getting fixated on a number.) However, most webmasters want to know how they rank, if only so they can gauge how the site's standing changes over time. Although you can't get the exact PageRank that Google uses in its search algorithms, you *can* get a simplified PageRank score that gives you a general idea of your website's performance. This simplified PageRank is based on the real PageRank, but it's updated just twice a year, and it only provides a value of from 1 to 10. (All things being equal, a website rated 10 will turn up much earlier in someone's search results than a page ranked 1.)

There are two ways to find your website's simplified PageRank. First, you can use the free Google Toolbar (www.google.com/toolbar), which snaps on to your browser window and provides a PageRank button. However, you need to explicitly enable this feature, as described at <http://tinyurl.com/64bjmtd>. A simpler approach is to use an unofficial PageRank-checking website, like the one shown in Figure 11-6. You simply type in your URL and click a button to get the simplified PageRank score.

If you want to delve into the nitty-gritty of *search engine optimization* (known to webmasters as SEO), consider becoming a regular reader of www.webmasterworld.com and www.searchengineland.com. You'll find articles and forums where webmasters discuss the good, bad, and downright seedy tricks you can try to get noticed.

Tip: It's possible to get too obsessed with search engine rankings. Here's a good rule of thumb: Don't spend more time trying to improve your search engine rank than you do improving your website. In the long term, the only way to gain real popularity is to become one of the best sites on the block.

**Figure 11-6:**

Use www.prchecker.info (or search Google for a different PageRank checker) to quickly rank any website. Here, it reports that www.prosetech.com scores a middle-of-the-road 4 out of 10.

Hiding from Search Engines

In rare situations, you might create a page that you *don't* want to turn up in a search result. The most common reason is because you posted some information that you want to share with only a few friends, like the latest Amazon e-coupons. If Google indexes your site, thousands of visitors could come your way, expecting to find a discount coupon and sucking up your bandwidth for the rest of the month. Another reason may be that you're posting something semi-private that you don't want other people to stumble across, like a story about how you stole a dozen staplers from your boss. If you fall into the latter category, be very cautious. Keeping search engines away is the least of your problems—once a site's on the Web, it *will* be discovered. And once it's discovered, it won't ever go away (see the box on page 299).

But you can do at least one thing to minimize your site's visibility or, possibly, keep it off search engines altogether. To understand how this procedure works, recall that search engines do their work in several stages. In the first, a robot program crawls across the Web, downloading sites. You can tell this robot not to index your site, or to ignore a portion of it, in several ways.

To keep a robot away from a single page, add the *robots* meta element to the page. Use the content value *noindex*, as shown here:

```
<meta name="robots" content="noindex" />
```

Remember, like all meta elements, you place this one in the *<head>* section of your HTML document.

Alternatively, you can use *nofollow* to tell robots to index the current page, but not to follow any of its links:

```
<meta name="robots" content="nofollow" />
```

If you want to block larger portions of your site, you're better off creating a specialized file called *robots.txt*, and placing it in the top-level folder of your site. The robot will check this file before it goes any further. The content inside the *robots.txt* file sets the rules.

If you want to stop a robot from indexing any part of your site, add this to the *robots.txt* file:

```
User-Agent: *  
Disallow: /
```

The User-Agent part identifies the type of robot you're addressing, and an asterisk represents all robots. The Disallow part indicates what part of the website is off limits; a single forward slash represent the whole site.

To rope off just the Photos subfolder on your site, use this (making sure to match the capitalization of the folder name exactly):

```
User-Agent: *  
Disallow: /Photos
```

To stop a robot from indexing certain types of content (like images), use this:

```
User-Agent: *  
Disallow: /*.gif  
Disallow: /*.jpeg
```

As this example shows, you can put as many Disallow rules as you want in the *robots.txt* file, one after the other.

Remember, the *robots.txt* file is just a set of *guidelines* for search engine robots; it's not a form of access control. In other words, it's similar to posting a "No Flyers" sign on your mailbox: It works only as long as advertisers choose to heed it.

Tip: You can learn much more about robots, including how to tell when they visit your site and how to restrict robots from specific search engines, at www.robots.txt.org.

UP TO SPEED

Web Permanence

You've probably heard a lot of talk about the ever-changing nature of the Web. Maybe you're worried that the links you create today will lead to dead sites or missing pages tomorrow. Well, there's actually a much different issue taking shape—copies of old sites that just won't go away.

Once you put your work on the Web, you've lost control of it forever. The corollary to this sobering thought is: Always make sure you aren't posting something that's illegal, infringes on copyright, is personally embarrassing, or could get you fired. Because once you put this material out on the Web, it may be there forever.

For example, imagine you accidentally reveal your company's trade secret for carrot-flavored chewing gum. A few weeks later, an excited customer links to your site. You realize your mistake, and pull the pages off your web server. But have you really contained the problem?

Assuming the Google robot has visited your site recently (which is more than likely), Google now has a copy of your old site. Even worse, people can get this *cached* (saved) copy from Google if they know about the *cache* keyword.

For example, if the offending page's URL is www.GumLover.com/newProduct.htm, a savvy Googler can get the old copy of your page using the search "cache:www.GumLover.com/newProduct.htm." (Less savvy visitors might still stumble onto a cached page by clicking the Cached link that appears after each search result in Google's listings.) Believe it or not, this trick's been used before to get accidentally leaked information, ranging from gossip to software license keys.

You can try to get your page out of Google's cache as quickly as possible using the remove URL feature at www.google.com/webmasters/tools/removals. But even if this works, you're probably starting to see the problem: You have no way to know how many search engines made copies of your work. Interested people who notice that you pulled information off of your site will hit these search engines and copy the details to their own sites, making it pretty near impossible to eliminate the lingering traces of your mistake. There are even catalogs dedicated to preserving old websites for posterity (see the Wayback Machine at www.archive.org).

The Google Webmaster Tools

If you're feeling a bit in the dark about your website's relationship with Google, you'll be happy to know that Google has a service that can help you out. It's called the Google Webmaster Tools, and it serves two purposes. First it provides information that can help you understand how Google indexes your pages. Second, it lets you tweak a few settings that govern how Google treats your site.

To use the Google Webmaster Tools, follow these steps:

1. Go to www.google.com/webmasters/tools and sign in with your Google account. If you don't have one, click "Create an account now."

A Google account lets you access a number of indispensable Google services, like Google Analytics (page 303), Gmail (Google's web-based mail service), Google Groups (Chapter 12), Google AdSense (Chapter 14), and Blogger (Chapter 13). Every webmaster has one.

- Once you sign in, you need to register your website. Click the “Add a site” button, enter in the full URL of your site (like `www.supermagicalpotatoes.co.uk`), and then click Continue.

This adds your website to your Google account. However, before you can actually view data for your site or manage its Google-related search settings, you need to prove that you are the legitimate owner of the site.

- Choose how you want to prove website ownership (see Figure 11-7).

Google lets you prove ownership several ways; it’s up to you to choose the option you want. Usually, the easiest choice is to add a Google-supplied meta element to your website’s home page, or you can upload a Google-supplied file to your website folder. In some situations, however, you might need a slightly more awkward approach. For example, if you use domain forwarding (page 59), Google will check for the uploaded file and meta element on the website that does the forwarding, which isn’t the website that holds your content. In this case, you need to contact your domain name provider and ask them to upload the verification file to your site. Or, you can verify your website by adding a DNS record.

The screenshot shows a Firefox browser window titled "Webmaster Central - Verify ownership - Mozilla Firefox". The address bar shows the URL `https://www.google.com/webmasters/verification/verification?hl=en&siteUrl=http://`. The main content area is titled "Verify ownership" and displays the status "Not Verified". A yellow box on the left labeled "Help with:" lists various verification methods: Understanding verification, Verifying with meta tag, Verifying with HTML, Verifying with DNS, Verifying with Google Analytics, and Verification errors. The "Verifying with meta tag" link is underlined, indicating it is the current method selected. Below this, instructions and a code snippet are provided:

There are several ways to prove to Google that you own `http://www.supermagicalpotatoes.co.uk/`. Select the option that is easiest for you.

- Add a DNS record to your domain's configuration
You can use this option if you can sign in to your domain registrar or hosting provider and add a new DNS record.
- Link to your Google Analytics account
You can use this option if your site already has a Google Analytics tracking code that uses the asynchronous snippet. You must be an administrator on the Analytics account.
- Upload an HTML file to your server
You can choose this option if you can upload new files to your site.
- Add a meta tag to your site's home page
You can choose this option if you can edit your site's HTML.

Instructions:

- Copy the meta tag below, and paste it into your site's home page. It should go in the <head> section, before the first <body> section.

```
<meta name="google-site-verification" content="a2KpI5QvmnvbaW6d0ZjvOc8rHDGxyLymwSoeXUzTY8" />
```
- Show me an example

2. Click verify below.
Leave the meta tag in place even after verification succeeds.

At the bottom are two buttons: "Verify" and "Do this later".

Figure 11-7:
For most people, the meta tag approach is the quickest way to integrate Google Webmaster Tools with your website. Google gives you a snippet of HTML, and you simply copy it to the <body> section of your home page (that's the index.html or index.htm page where web visitors start out).

Note: Rest assured, no matter what verification choice you pick, it won't affect the way your website works. In fact, Google recommends that you don't remove the meta element you added (or delete the verification file you uploaded). That way, you won't need to verify your ownership to Google again.

4. Once you prove site ownership, click Verify.

Usually, Google can successfully verify your site mere seconds after you make the appropriate change (whether that's adding a meta element or uploading a verification file). Once Google knows that you're the real owner, it refreshes the page and adds a pile of management links to the left side (see Figure 11-8). From this point on, whenever you log in to the Google Webmaster Tools page you'll be able to use all of Google's tools to review and manage your site.

The Google Webmaster Tools let you look at your website through the eyes of Google. You choose what you want to do by clicking one of the links in the Dashboard section on the left side of the page. Here are some of the possibilities:

- **Site configuration > Sitemaps.** This section helps you build a *sitemap*—a special file that describes the structure of your site and the files in it. You can submit your sitemap to Google and other search engines so they know what to index. This is particularly useful if you have pages that Google might ordinarily miss, like standalone pages (those not linked to other pages).
- **Site configuration > Crawler access.** This section helps you create a *robots.txt* files, which hides a portion of your site from nosy search engines, as explained on page 297.
- **Site configuration > Sitelinks.** Sitelinks are the quick-access links that appear underneath some search listings and let visitors jump straight to a specific page in the website. Unfortunately, only Google can decide to reward your website with sitelinks (based on the popularity and structure of your site). If Google does give you sitelinks, however, you can use this section to remove any that aren't appropriate.
- **Site configuration > Change of address.** Moving to a new URL? This section helps you redirect Google to your new home, so you can take your web traffic with you.
- **Site configuration > Settings.** This section lets you tweak a few miscellaneous settings for how Google treats your site. The most important is "Geographic target." Make sure you set it to the country you live in so that your site turns up in the right regional searches.
- **Your site on the web > Search queries.** Here, you get information about the searches that lead Googlers from the search engine to your website (see Figure 11-8).
- **Your site on the web > Links to your site.** This option shows you which websites link to yours. It's a useful way to see who's paying attention to your content.

- **Diagnostics.** This section warns you about any problems Google encounters as it indexes your site, like incorrect metadata (page 287), pages that it couldn't access (and therefore couldn't index), or evil malware lurking on your server.

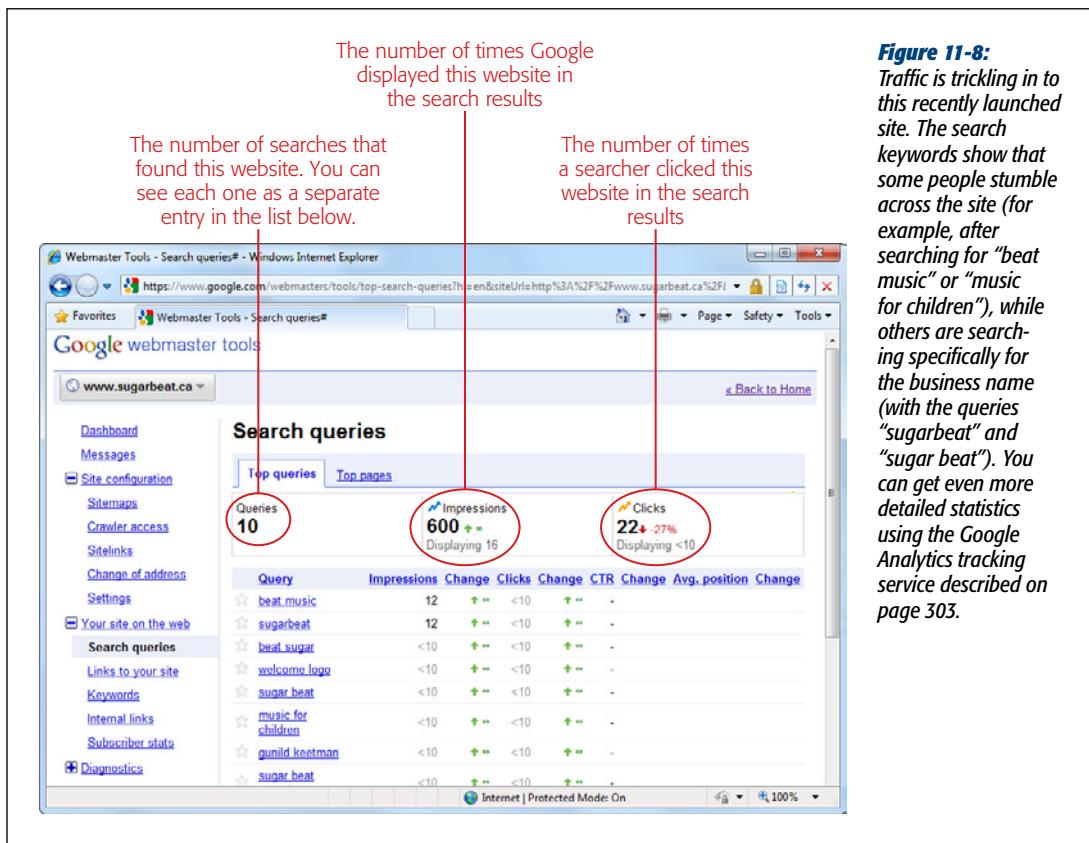


Figure 11-8:
Traffic is trickling in to this recently launched site. The search keywords show that some people stumble across the site (for example, after searching for “beat music” or “music for children”), while others are searching specifically for the business name (with the queries “sugarbeat” and “sugar beat”). You can get even more detailed statistics using the Google Analytics tracking service described on page 303.

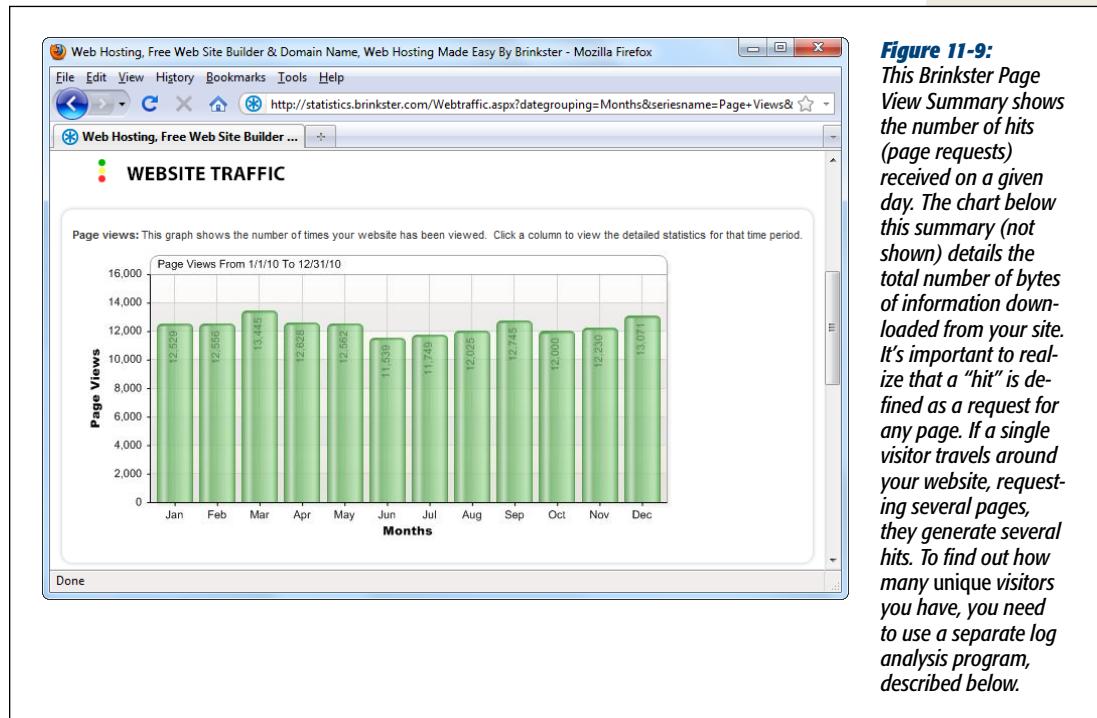
Most serious web designers eventually check out their sites with Google Webmaster Tools. If nothing else, you can use it to make sure everything is running smoothly—in other words, that Google can get to your pages, that its automated search robot returns frequently to check for new content, and that the robot reviews all the pages you have to offer.

Tracking Visitors

As a website owner, you'll try a lot of different tactics to promote your site. Naturally, some will work while others won't—and you need to keep the good strategies and prune those that fail. To do this successfully, you need a way to assess how your site performs.

Almost every web hosting company (except free web hosts) gives you some way to track the number of visitors to your site (see Figure 11-9). Ask your host how to use

these tools. Usually, you need to log on to a “Control Panel” or “My Account” section of your host’s site. You’ll see a variety of options there; look for an icon labeled “Site Counters” or “Web Traffic.”



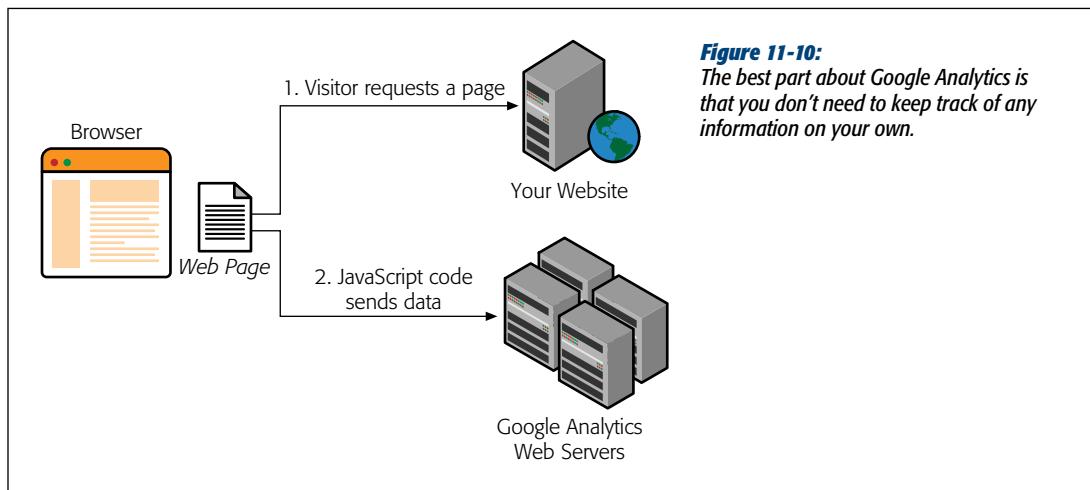
With more high-end hosting services, you often have more options for viewing your site’s traffic statistics. Some hosts provide the raw *web server logs*, which store detailed, blow-by-blow information about every one of your visitors. This information includes the time a visitor came to your site, their IP addresses (page 52), their browser type, what site referred them to you, whether they ran into an error, what pages they ignored, what pages they loved, and so on. To make sense of this information, you need to feed it into a high-powered program that performs *log analysis*. These programs are often complex and expensive. An equally powerful but much more convenient approach is to use the Google Analytics tracking service, described next.

Understanding Google Analytics

In 2005, Google purchased Urchin, one of the premium web tracking companies. They transformed it into *Google Analytics* and abolished its hefty \$500-a-month price tag, making it free for everyone. Today, Google Analytics just might be the best way to see what’s happening on any website, whether you’re building a three-page site about dancing hamsters or a massive compendium of movie reviews.

Google Analytics is refreshingly simple. Unlike other log analysis tools, it doesn't ask you to provide server logs or other low-level information. Instead, it tracks all the information you need *on its own*. It stores this information indefinitely, and lets you analyze it any time with a range of snazzy web reports.

To use Google Analytics, you need to add a snippet of JavaScript code to every web page you want to track (usually, that's every page on your site). Once you get the code in place, everything works seamlessly. When a visitor heads to a page on your site, the browser sends a record of the request to Google's army of monster web servers, which store it for later analysis. The visitor doesn't see the Google Analytics stuff. Figure 11-10 shows you how it works.



Note: Remember, JavaScript is a type of mini-program that runs inside virtually every browser in existence. Chapter 14 provides an introduction to JavaScript.

Using this system, Google Analytics collects two kinds of information:

- **Information about your visitors' browsers and computers.** Whenever a browser requests a page, it supplies a basic set of information. This includes the type of browser it is, the features it supports, and the IP address of the computer it connects through (an *IP address* is a numeric code that uniquely identifies a computer on the Internet). These details don't include the information you really want—for example, there's no way to find out personal details like names or addresses. However, Google uses other browser information to infer additional details. For example, using the IP address, it makes an educated guess about your visitor's geographic location.

- **Visitor tracking.** Thanks to its sophisticated tracking system, Google Analytics can determine more interesting information about a visitor's patterns. Ordinarily, if a visitor requests two separate pages from the same site, there's no way to establish whether those requests came from the same person. However, Google uses a *cookie* (a small packet of data stored on a visitor's computer) to uniquely identify each visitor. As a result, when visitors click links and move from page to page, Google can determine their navigation path, the amount of time they spend on each page, and whether they return later.

Google Analytics wouldn't be nearly as useful if it were up to you to make sense of all this information. But as you'll see, Google not only tracks these details, it provides reports that help you figure out what the data really means. You generate the reports using a handy web screen menu, and you can print them out or download them for use it in another program, like Excel, to do further analysis.

Signing Up for Google Analytics

Signing up for Google Analytics is easy:

1. Head over to www.google.com/analytics, and click the Sign Up Now link.

Google Analytics is yet another one of many services you can access with a single Google account. If you don't have one, you'll be prompted to create one by providing your email address and password.

2. Fill in the information about your website.

The two pieces of information you need to supply are:

- The URL for the site you want to track (for example, *www.supermagicalpotatoes.co.uk*). A Google Analytics account can track as many sites as you like, but for now start with just one.

Your time zone. This lets Google Analytics synchronize its clock with yours.

3. Click Finish.

After the sign-in process, Google gives you a box with the JavaScript code you need to add to your pages to start tracking visitors (see Figure 11-11). The next section tells you how.

4. Add the tracking code to your web pages.

When you add Google's code to a page, put it at the very end of the `<head>` section, just before the closing `</head>` tag. Here's an example of where it fits in a typical web page:

```
<!DOCTYPE>
```

```
<html>
<head>
  <title>Welcome</title>
  <!-- Put the analytics code here. -->
</head>
```

```
<body>
...
</body>
</html>
```

Be advised, however, that Google Analytics used a different approach for several years, which put slightly different tracking code after the `</body>` tag. So if you look at another page that uses Google Analytics, you might find the tracking code in a different place.

Figure 11-11:
The Google Analytics code is lean and concise, requiring just a few lines. It links to a file on Google's web servers to get the real tracking code. Select all the code displayed, and then copy it to your clipboard (you do this in most browsers by right-clicking the selected text, and then choosing Copy).

Tip: For best results, copy the tracking code to every page in your site.

5. Upload the new version of your web pages.

Once you change all your pages, make sure to upload them to your server. Only then can Google Analytics start tracking visitors. The tracking features won't work when you run the pages from your own computer's hard drive.

Now, it's a waiting game. Within 24 hours, Google Analytics has enough information about recent visitors to provide its detailed reports.

Examining Your Web Traffic

As a registered Google Analytics user, you can log in to read reports and make sure everything's running smoothly. If you haven't already done so, now's the time to head to www.google.com/analytics.

When you first log in, you see the Analytics Settings page shown in Figure 11-12. Using this page, you can add new websites to track or configure existing ones.

Figure 11-12:
This sample account tracks two websites, but only one is successfully collecting data (as indicated by the checkmark in the status column). You can use the View Reports link to start reviewing reports, the Edit link to change the information you supplied for your website, and the Delete link to remove your profile altogether.

To determine whether your website's tracking code works, hover over the icon in the Status column. Here's how to interpret the text that you'll see:

- **Receiving Data** indicates that all's well. Your visitors are going from page to page under the watchful eye of Google Analytics.
- **Waiting For Data** indicates that Google's JavaScript code is running on your pages, but the information needed for a report isn't available yet. Usually, you see this for the first 6 to 12 hours after you register a new site.
- **Tracking Data Not Installed** indicates that Google isn't collecting any information. This could be because you need to wait for visitors to hit your site, or it could suggest you haven't inserted the correct JavaScript tracking code.

If you see Receiving Data, congratulations! Your reports are ready and waiting. Click the View Reports link next to your website URL to continue on to the Dashboard (Figure 11-13).

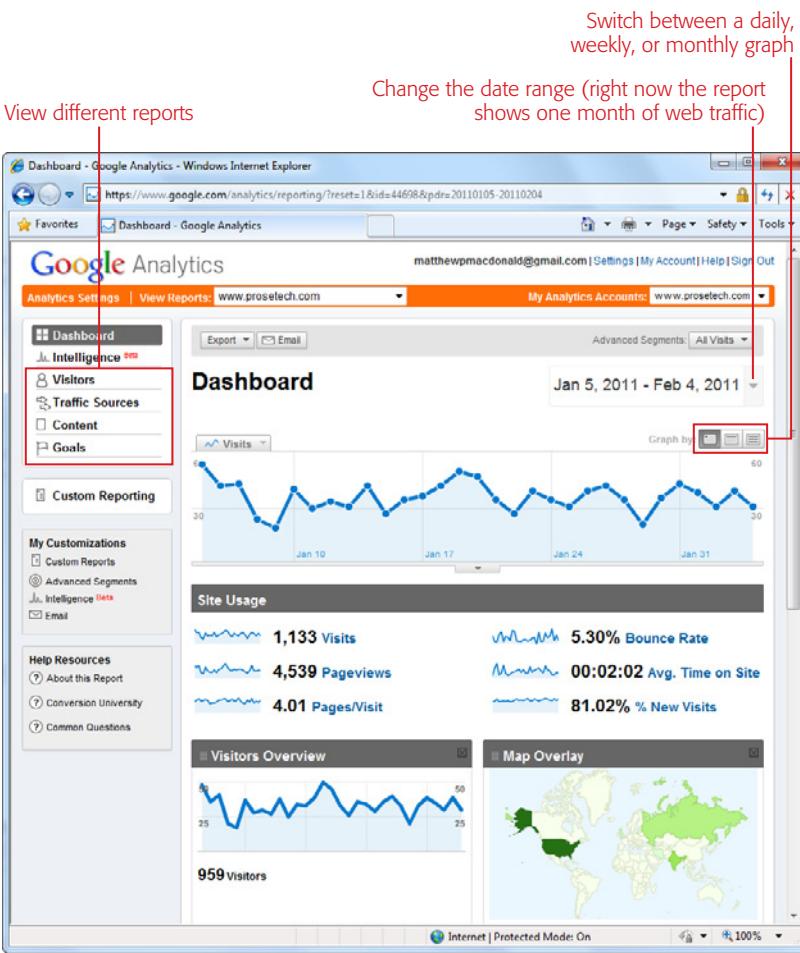


Figure 11-13:
The Dashboard displays a number of basic charts that detail your website's vital signs. The menu on the left lets you browse to a variety of more specialized reports.

The information the Dashboard displays can be a little overwhelming. To give you a better sense of what's going on, the following sections break down the key points, one chart at a time.

Graph of visits

At the top of the page, the graph of visits (Figure 11-14) shows the day-by-day popularity of your site over the last month. This count doesn't say anything about how many pages the average visitor browsed or how long she lingered. It simply records how many different people visited your site. Using this chart, you can get a quick sense of the overall uptrend or downtrend of your web traffic, and you can see if it rises on certain days of the week or around specific dates.



Figure 11-14:
To get the specific value for a data point, point to it. For example, this chart clearly indicates a modest 51 visits on January 19.

With a few clicks, you can change this chart to show *pageviews*, the count of how many web pages all your visitors viewed. For example, if an eager shopper visits your Banana Painting e-commerce store, checks out several enticing products, and completes a purchase, Google Analytics might record close to a dozen pageviews but only a single visit. The number of pageviews is always equal to or greater than the number of visits; after all, each visit includes at least one pageview. To see pageviews, click the down arrow next to the word “Visits” in the top-left corner of the chart, and then choose Pageviews from the list of options.

Tip: Remember, you can look at the data Google Analytics collects over a different range of dates using the date box in the top-right corner of the Dashboard, as identified in Figure 11-13.

Site Usage

The Site Usage section is crammed with key statistics (Figure 11-15). Details include:

- The total number of visits for the selected date range (initially, it's the month leading up to the current day).
- The total number of pageviews.
- The average number of pages a visitor reads before leaving your site.
- The bounce rate. A *bounce* occurs when a visitor views only one page—in other words, they get to your site through a specific page, and then depart without browsing any further. A bounce rate of 5 percent tells you that 5 percent of your visitors leave immediately after they arrive. Bounces are keenly important to webmasters because they indicate potential lost visitors. If you can identify what's causing a big bounce, you can capture a few more visitors.
- The average time a visitor spends on your site before browsing elsewhere.
- The percentage of new visits. For example, a rate of 81 percent indicates that 19 percent of your traffic is repeat business, and 81 percent are new visitors. Both types of visitor are important to your website health.



Figure 11-15:
The Site Usage numbers are the most important indicator of your site's web health. You can click any of these stats to see a separate report with more detail.

Note: There are some types of repeat visitors that Google won't correctly identify. For example, if a repeat visitor uses a different computer, a different browser, or logs into his computer with a different user name, they'll appear to be a new visitor. For these reasons, the number of repeat visitors may be slightly underreported.

Map Overlay

The Map Overlay section gives you a fascinating look at where your visitors are located on the globe. Google Analytics divides the map into countries, and colors them different shades of green—the darker the color, the more popular your website in that country.

The Map Overlay gets even more interesting if you use the City view, shown in Figure 11-16. To see it, click the “view report” link under the Map Overlay on the Dashboard. A new page appears with a larger version of the map and a detailed breakdown in a table underneath. Look for the Detail Level heading under the map, and click City.

FREQUENTLY ASKED QUESTION

Does Google Really Know Where I Live?

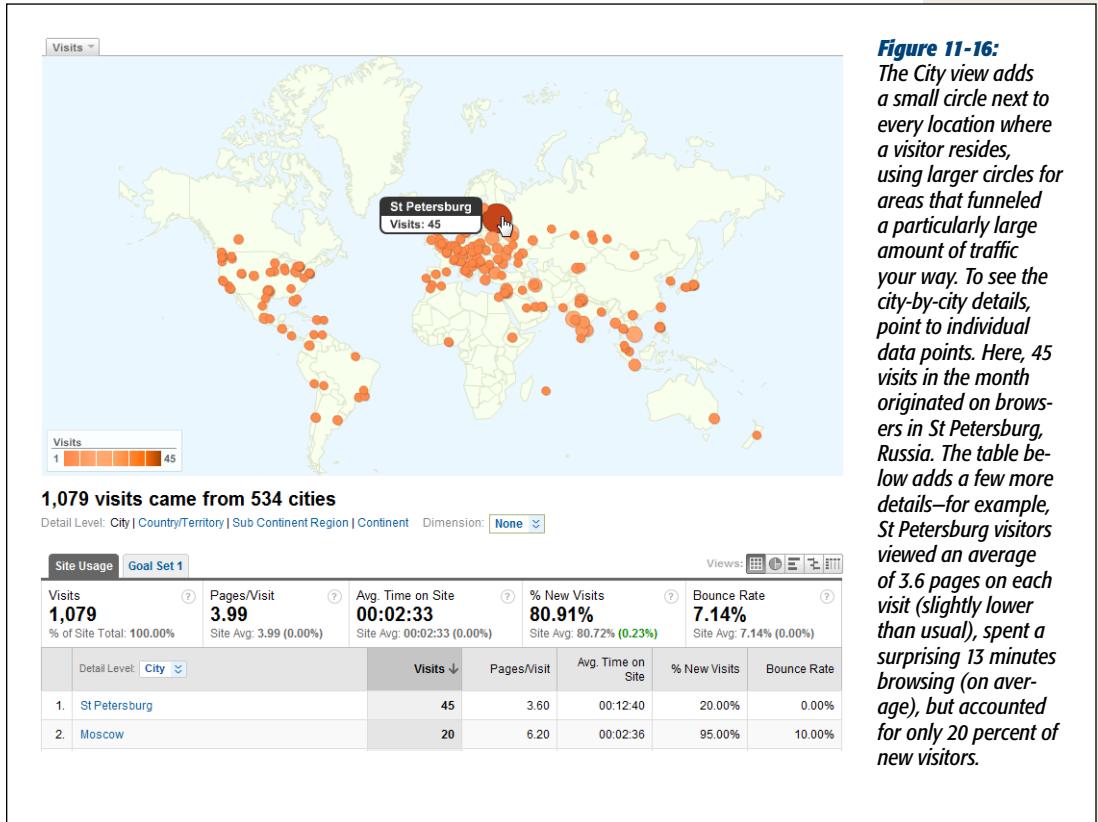
How accurate is the location data Google supplies?

Being able to determine the location of your visitors is a powerful tool. After all, if you know your Graceland Vacation site is absurdly popular in Japan, you might consider accepting payments in Yen, translating a few pages, or adding some new pictures. But Google's geographic locating service isn't perfect. In fact, there are two weaknesses:

- The location service is based on a visitor's ISP (Internet Service Provider), not the actual visitor herself. In many cases, the ISP is located in a different area than the visitor's own computer.

- ISPs economize by pooling their traffic together and dumping it onto the Internet at a central location. This means that even if your visitor and her web server are in a specific city, the computer that connects these visitors to the Internet might be somewhere else.

As a general rule of thumb, the geographic information that Google uses is likely to be close to reality, but not exact. It's highly likely that the country is correct, but the specific city may not match that of your visitor.

**Figure 11-16:**

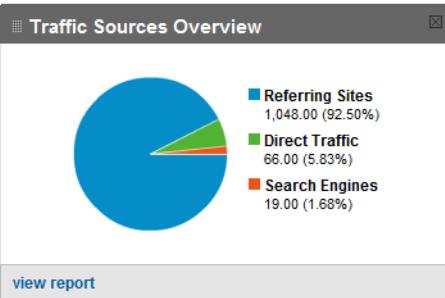
The City view adds a small circle next to every location where a visitor resides, using larger circles for areas that funneled a particularly large amount of traffic your way. To see the city-by-city details, point to individual data points. Here, 45 visits in the month originated on browsers in St Petersburg, Russia. The table below adds a few more details—for example, St Petersburg visitors viewed an average of 3.6 pages on each visit (slightly lower than usual), spent a surprising 13 minutes browsing (on average), but accounted for only 20 percent of new visitors.

Traffic Sources Overview

The Traffic Sources Overview chart identifies how visitors get to your site. It has three slices:

- **Referring Sites.** This slice counts visitors that arrive through websites that link to yours.
- **Direct Traffic.** This slice counts visitors that type your URL directly into a browser or use a bookmark to make a return visit.
- **Search Engines.** This slice counts visitors who come to your site through a search in Google or other search engine.

Figure 11-17 shows a closer look.

**Figure 11-17:**

Over the past month, 92 percent of this site's visitors came from referring sites, while 6 percent came directly to the site (by typing in its URL). A meager 2 percent found their way through a search engine, which suggests you might want to consider investing more time in search engine optimization (page 296).

The Traffic Sources Overview chart provides a good high-level look at how your visitors get to your site. To see more details, click the “view report” link underneath, or use the Traffic Sources menu on the left. You can explore tables that break down the Referring Sites and Search Engines slices. For the Referring Sites slice, you see exactly which websites send you traffic, which is a great tool for quickly identifying your most successful web partnerships. For the Search Engines slice, you can review the search keywords that lead to your site, which lets you determine what your visitors are looking for (and possibly diagnose why they left unhappy).

Content Overview

As every webmaster knows, all pages aren't created equal. Some might command tremendous interest while others languish, ignored. The Content Overview section lists top-performing pages (Figure 11-18).

Content Overview		
Pages	Pageviews	% Pageviews
/prosetech/BooksNET.htm	1,482	32.65%
/prosetech/	1,214	26.75%
/prosetech/Classes.htm	353	7.78%
/prosetech/ProWPFinC2010.htm	240	5.29%
/prosetech/About.htm	186	4.10%
view report		

Figure 11-18:

In this site, BooksNET.htm is the clear winner of the Most Popular Page award, with 32 percent of the total website traffic.

However, simply determining which pages your visitors view the most is not enough. Some pages are extra important because of their ability to *attract* visitors. For example, the page of Member Photos on your International Nudist site might attract large volumes of visitors who then stick around to check out your personalized coffee cups, clothing, and memorabilia. A good reporting tool also shows you where visitors *enter* your site, so you know what pages are attention-getters that lure traffic. Almost as important are the pages that mark the end of a visit. They may indicate a problem, like a page that's slow to download, doesn't work correctly in some browsers, or just plain irritates people. To get this sort of detailed information in Google Analytics, click the "view report" button in the bottom of the Content Overview section, or use the Content menu on the left side of the page.

Note: There are many more reports you can explore in Google Analytics, and many more ways to slice and dice its results to come up with conclusions about your web traffic. In fact, entire books have been written about the fine art of analyzing website performance. However, the five charts explained above get you started with great insight into how your site is doing.

Website Promotion

The best website in the world won't do you much good if it's sitting out there all by its lonesome self. For your site to flourish, you need to attract new visitors, and then keep them flocking back for more.

In the previous chapter, you started out on this path. You learned how to get your site noticed in a web search, and how to keep track of the number of visitors that stop by. But a search listing, on its own, isn't enough to grow a brand-new site into a thriving web destination. For that, you need a range of promotional tactics, from sharing links to buying ad space. Contrary to what you might expect, this sort of ground-roots promotion might bring more traffic to your site than high-powered search engines like Google do.

In this chapter, you'll learn some of the best techniques for website promotion. You'll also see how to build a *sticky* site—one that not only attracts new faces, but encourages repeat visitors. To pull this off, you need to transform your website into a web *community*. Your visitors will need a way to interact with your content and talk to you and each other, and you can use Facebook, Twitter, blogs, and groups to make this happen.

Note: The tasks you perform in most of this book—crafting web pages, formatting them with style sheets, uploading your finished site—are relatively straightforward. They might take some time and effort, but when you're done, you *know* you're done. The tasks you'll tackle in this chapter—promoting your website and building a community around it—are not so well defined. They require continuous work, and it may take a lot of thankless slogging before your website traffic starts to grow.

Spreading the Word

To promote your website well, you need to master many skills. Good website promotion ranges from old-fashioned advertising to search engine magic.

In the following sections, you'll pick up several fundamental techniques. You'll start thinking about how to promote yourself in the right places, take a quick look at services like Google Places and Google AdWords, and plan a strategy that encourages repeat visits.

Shameless Self-Promotion

To get your website listed on many of the Web's most popular sites, you need to fork over some cold, hard cash. However, some of the best advertising doesn't cost anything: Look for sites where you can promote yourself *and* contribute at the same time.

For example, if you create the website *www.HotComputerTricks.com*, why not answer a few questions on a computing newsgroup, discussion board, or Facebook page? Openly promoting your site is considered tactless, but there's nothing wrong with dispensing some handy advice and following it up with a signature that includes your URL.

Here's an example of how you can answer a poster's question and put in a good word for yourself at the same time:

Jim,

The problem is that most hard drives will fail when submerged in water. Hence, your fishing computer idea won't work.

Sasha Mednick

www.HotComputerTricks.com

An answer posting is much better than sending an email directly to the original poster because on a popular site, hundreds of computer aficionados with the same question will read your post. If even a few decide to check out your site, you've made great progress.

If you're very careful, you might even get away with something that's a little more explicit:

Jim,

The problem is that most hard drives will fail when submerged in water. Hence, your fishing computer idea won't work. However, you might want to check out my home-made hard-drive vacuum enclosure (*www.HotComputerTricks.com*), which I developed to solve the same problem.

Sasha Mednick

www.HotComputerTricks.com

Waning: This maneuver requires a very light touch. The rule of thumb is that your message should be well intentioned. Only direct someone to your site if there really is something specific there that addresses the question.

Some sites let you post tips, reviews, or articles. If that's the case, you can use a variation of the technique above. Remember, dispense useful advice, and then follow it up with a byline at the end of your message. For example, if you submit a free article that describes how to create your groundbreaking vacuum enclosure, end it with this:

Sasha Mednick is a computer genius who runs the first-rate computing site www.HotComputerTricks.com.

Promotion always works best if you believe in your product. So make sure you have relevant high-quality content on your site before you boast about it. Don't ever send someone to your site based on content you plan to add (someday).

Tip: If you're a business trying to promote a product, you'll get further if you recruit other people to help you spread the word. One excellent idea is to look for influential *bloggers*—people who create websites with the personal posting format you'll learn about in Chapter 13. For example, if you're trying to sell a new type of fluffy toddler towel pajamas, hunt down popular people with blogs about parenting. Then, offer them some free pajamas if they'll offer their thoughts in a blog review. This sort of word-of-mouth promotion can be dramatically more successful in the wide-reaching communities of the Web than it is in the ordinary offline world.

Google Places

In the previous chapter, you learned how your site can become a result in someone's web search. This is a critical way to reel in new people, and many webmasters spend their late-night hours obsessing about page ranking and search keywords.

However, if you're running a business or organization that has a physical presence in the real world (not just the virtual one), there's an easy tune-up that can make your site stand out from the crowd. In fact, you've probably seen it before, if you've ever used Google to hunt down a local business (Figure 12-1).

Clearly, this listing has more presence than the average search listing. It takes up more space, provides more information, and makes the business seem more professional. When visitors see a listing like this, they feel that they've come across a real, established business.

The service that makes this happen is called Google Places. Happily, you can sign up for Google Places and have your own beefed-up search listing for free. There are just two things to keep in mind:

- **You need to provide your location's mailing address.** To verify that your business exists at the location you say it does, Google will mail you a confirmation code. Once you enter that code, your Google Places listing becomes active. If

you don't have a physical mailing address you can use (such as your home), or you don't want it to appear on Google, the Google Places service isn't for you.

- **The enhanced listing only appears for certain searches.** Google's goal is to show the enhanced listing when a web searcher specifically looks for your business. So if someone types your business name or street address into Google, they're likely to get the enhanced page. But if someone simply enters a few keywords that match the content on your website, Google displays only an ordinary search listing.

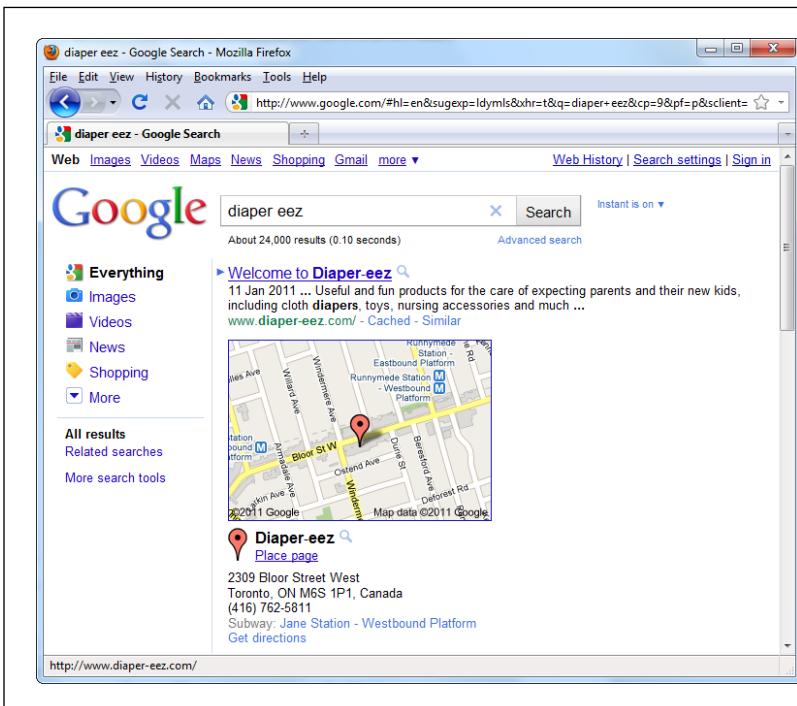


Figure 12-1: The first result in this search is an extended listing that includes a map, address, and a link for directions. Click on the “Place page” link and you’ll get a full business listing, with pictures.

The screenshot shows a Google search results page for the query "diaper eez". The top result is an extended listing for "Welcome to Diaper-eez" located at 2309 Bloor Street West, Toronto, ON M6S 1P1, Canada. The listing includes a map showing the location on Bloor St W between Roncesvalles Ave and Dufferin Ave, with a red marker indicating the exact address. Below the map, the business name "Diaper-eez" is listed with a "Place page" link. The "Place page" link leads to a detailed business profile page on Google's servers.

To create your own Google Places listing, head to <http://places.google.com/business>. If you have a Google account (used for other services like Gmail and Google Webmaster Tools), sign in now. Otherwise, click “Sign up now” to create one.

After you log in, you can add your business listing. You need to fill out a single-page form of business information that includes your address, phone number, website, a short description, and category. Optionally, you can upload pictures (which is always a good idea, because people will see them on your Google Places page), add YouTube videos, and specify operating hours. When you finish, you submit the form to Google. Remember, your listing won't appear automatically, because Google needs to send you a confirmation code by mail. Type in the code to activate your Google Places page. It's then just a matter of time before your enhanced listing starts showing up in web searches.

Google AdWords

As a web-head, you've no doubt seen several lifetimes' worth of flashing messages, gaudy banners, and invasive pop-ups, all trying to sell you some hideously awful products. It probably comes as no surprise to learn that these types of ads aren't the way to promote your site—in fact, they're more likely to alienate people than entice them. However, there are respectable paid placements that can get your site in front of the right readers, at the right time, and with the right amount of tact. One of the best is AdWords (<http://adwords.google.com>), Google's insanely flexible advertising system.

The idea behind AdWords is that you create text ads that Google shows alongside its regular search results (see Figure 12-2). The neat part is that Google doesn't show the ads indiscriminately. Instead, you choose the search keywords you want your ad associated with.

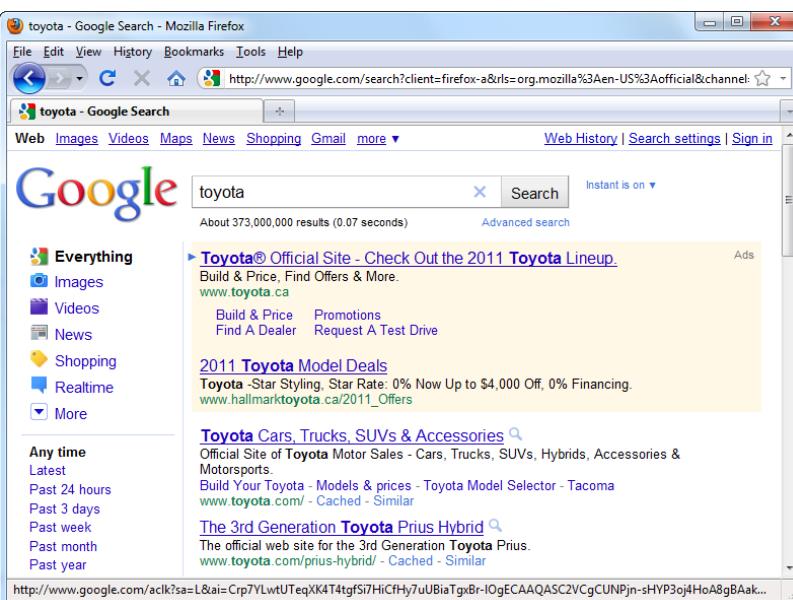


Figure 12-2: To see AdWords in action, try searching for a name brand like Toyota. You see a section clearly marked Ads on the right side of the search results, or just above the search results in a yellow shaded box (as in this example).

The screenshot shows a Google search results page for the query "toyota". The search bar contains "toyota" and the search button is labeled "Search". Below the search bar, it says "About 373,000,000 results (0.07 seconds)" and "Instant is on". The left sidebar includes links for Everything, Images, Videos, News, Shopping, Realtime, and More, along with filters for Any time, Latest, Past 24 hours, Past 3 days, Past week, Past month, and Past year. The main search results include several ads and organic search results. One ad is highlighted with a yellow background and the word "Ads" above it. Another ad is located above the organic results. Organic results include links to the Toyota official site, Toyota Model Deals, Toyota Cars, Trucks, SUVs & Accessories, and The 3rd Generation Toyota Prius Hybrid.

The nice (and slightly confusing) part about AdWords is that you *bid* for the keywords you want to use. For example, you might tell Google you're willing to pay 25 cents for the keyword "food." Google takes this into consideration with everyone else's bids, and displays the higher bidders' ads more often. But Google isn't out to rip anyone off, and it charges you only the going rate for your keyword, regardless of how much you told Google you're willing to pay. And Google doesn't charge you anything to simply display your ad on a search results page. It charges you only when someone clicks on your ad to get to your site.

By this point, you might be getting a little nervous. Given the fact that Google handles hundreds of millions of searches a day, isn't it possible for a measly one-cent bid to quickly put you and your site into bankruptcy? Fortunately, Google has the solution for this, too. You just tell Google how much you're willing to pay per day. Once you hit your limit, Google stops showing your ad.

Interestingly, the bid amount isn't the only factor that determines how often your ad appears. Popularity is also important. If Google shows your ad over and over again and it never gets a click, Google realizes that your ad just isn't working, and lets you know with an automatic email message. Google may then start showing your ad significantly less often or stop showing it altogether, until you improve it.

You can also limit your ads to people in certain geographic regions (for example, in a specific country or city). This way, you don't waste money showing ads for your kiddie hair salon to people in Tunisia. This sort of geographic targeting makes AdWords equally effective for both local and global promotion.

AdWords can be competitive. To have a chance against all the AdWords sharks, you need to know how much a click is worth to your site. For example, if you sell monogrammed socks, you need to know what percentage of visitors actually *buy* something (the *conversion rate*) and how much they're likely to spend. A typical cost-per-click hovers around 75 cents, but there's a wide range. At last measure, the word *lawyer* rung in at \$8.53, while the keyword combination *llama care* could be had for a song—a mere 5 cents. (And in recent history, law firms have bid “mesothelioma”—an asbestos-related cancer that could become the basis of a class-action lawsuit—up close to \$100.) Before you sign up with AdWords, it's a good idea to conduct some serious research to find out the recent prices of the keywords you want to use.

Note: You can learn more about AdWords from Google's AdWords site (<http://adwords.google.com>). For a change of pace, go to www.literature.com/adwords for a story about an artist's attempt to use AdWords to distribute poetry, and why it failed.

Return Visitors

Attracting fresh faces is a critical part of website promotion, but novice webmasters often forget something equally important—return visitors. For a website to become truly popular, it needs to attract visitors who return again and again. Many a website creator would do better to spend less time trying to attract new visitors and more time trying to keep the current flock.

If you're a marketer, you know that a customer who comes back to the same store three or four times is a lot more likely to make a purchase than someone who's there on a first visit. These regulars are also more likely to get excited and recruit their friends to come and take a look. This infectious enthusiasm can lead more and more people to your website's virtual doorstep. The phenomenon is so common it has a name: the *traffic virus*.

Note: Return visitors are the ultimate measuring stick of website success. If you can't interest someone enough to come back again, your website's just not fulfilling its destiny.

So how does your website become a favorite stopping point for web travelers? The old Internet adage says it all—*content is king*. Your site needs to be chock full of fascinating must-read information. Just as important, this information needs to change regularly and noticeably. If you update information once a month, your website barely has a pulse. But if you update it two or more times a week, you're ready to flourish.

Never underestimate the importance of regular updates. It takes weeks and months of up-to-date information to create a return visitor. However, one dry spell—say, three months without changing anything more than the color of your buttons—doesn't just stop attracting newcomers, it can kill off your current roster of return visitors. Savvy visitors immediately realize when a website's gone stale. They have much the same sensation you feel when you pull out a once-attractive pastry from the fridge and find it's as hard as igneous rock. You know what happens next: Toss the pastry away, clear out the website bookmarks, and move on.

Note: Signs of a stale site include old-fashioned formatting, broken links, and references to old events (like a Spice Girls CD release party or a technical analysis of why Florida condos are an ironclad investment).

The other way to encourage return visitors is to build a community. Discussion forums, promotional events, and newsletters are like glue. They encourage visitors to feel like they're participating in your site and sharing your web space. If you get this right, hordes of visitors will move in and never want to leave.

GEM IN THE ROUGH

Favorite Icons

One of your first challenges in promoting your site is getting visitors to add your site to their browser bookmarks. Bookmarking, however, is not enough to guarantee a return visit. Your website also needs to be fascinating enough to beckon from the bookmark menu, tempting visitors to come back. If you're a typical web traveler, you regularly visit only about 5 percent of the sites you bookmark.

One way to make your site stand out from the crowd is to change the icon that appears in visitors' bookmarks or favorites menu (an icon technically called a *favicon*). This trick works in any modern browser (see Figure 12-3).

To create a favicon, add an icon file to the top-level folder of your website, and make sure you name it *favicon.ico*. The best approach is to use a dedicated icon editor, because it lets you create both a 16-pixel × 16-pixel icon and a larger 32-pixel × 32-pixel icon in the same file. Browsers use the smaller icon in their bookmark menus, and computers display the larger version when visitors drag the favicon to their desktop. If you don't have an icon editor, just create a bitmap (a .bmp file) that's exactly 16 pixels wide and 16 pixels high. To get an icon editor, visit a shareware site like www.download.com.

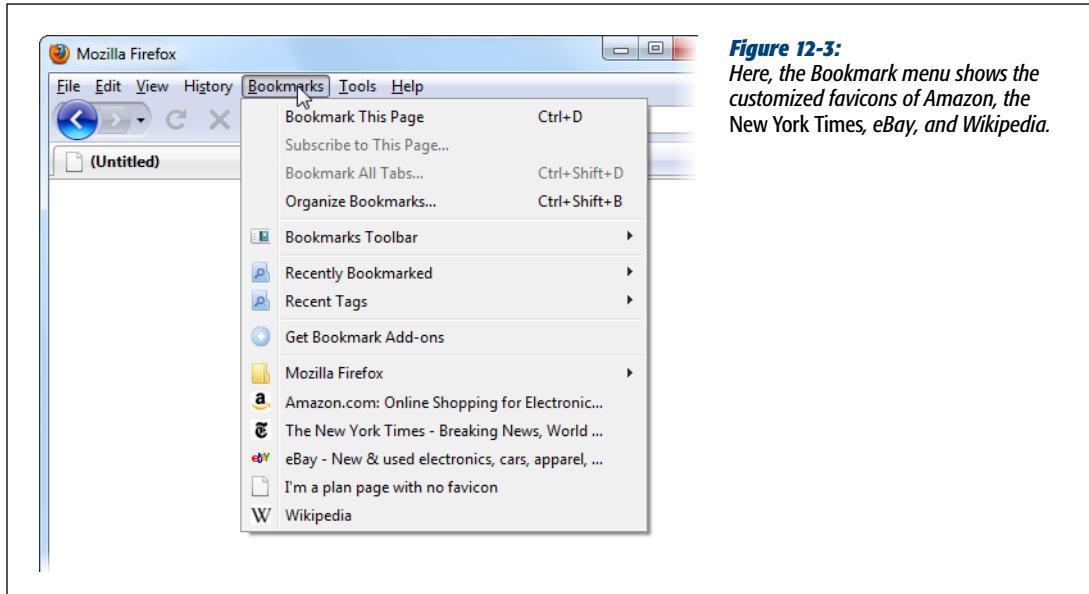


Figure 12-3:

Here, the *Bookmark* menu shows the customized favicons of Amazon, the New York Times, eBay, and Wikipedia.

Transforming a Site into a Community

The Web is the crowded home of several million people, so when you put your website online, it doesn't just drop into a vacuum. Instead, it takes center stage in front of an audience that's always interested and often opinionated.

For your site to really fit in with the rest of the Web, you need to interact with your visitors. The idea of dialogue—back-and-forth communication among peers—is hard-wired into the Internet's DNA. Get it right and people won't just be talking about you—they'll be promoting you to friends, getting to know you better, and putting you in the public eye.

To make this magic happen, you need a bit of planning, a willingness to promote yourself (almost to the point of embarrassment), and a smattering of social media skills. In the following sections, you'll learn how to do all this.

Fostering a Web Community

How do you start transforming your website into a web community? The first trick is to change your perspective, so that you plan your website as a meeting place instead of as just a place to vent your (admittedly brilliant) thoughts. Here are a few tips to help you get in the right frame of mind:

- **Clearly define the purpose of your site.** For example, the description “www.BronteRevival.com is dedicated to bringing Charlotte Bronte fans together to discuss and promote her work” is more community-oriented than “www.BronteRevival.com”

.BronteRevival.com contains information and criticism of Charlotte Bronte's work." The first sentence describes what the site aims to do, while the second reflects what it contains, thereby limiting its scope. Once you define a single-sentence description, you can use it in your description meta element (page 287) or in a mission statement on your home page.

- **Build gathering places.** No one wants to hang around a collection of links and static text. Jazz up your site with discussion forums or chat boards, where your visitors can kick up their heels. You'll learn how to get these bits in place later in this chapter.
- **Give your visitors different roles.** Successful community sites recognize different levels of contribution. At one extreme, the right people can grow into leadership roles and even coordinate events, newsletters, discussion groups, or portions of your site. At the other extreme are visitors who are happiest lurking in the background, watching what others do. There are different ways to recognize individual contributions—some sites use a personal feedback rating system that adds gold stars (or some other sort of icon) next to a person's name. Another approach is to give certain visitors more power, like the ability to manage members in a Google group (page 329).
- **Advertise new content before and after you add it.** To get visitors coming back again and again, you need lots of new content. But new content on its own isn't enough—you need to build up visitors' expectation of new content so they know enough to return, and you need to clearly highlight the new material so guests can find it once they do come back. To help this work smoothly, try adding links on your first page that lead to newly added content, along with a quick line or two about upcoming content you plan to add and concrete information about when it'll be there.
- **Introduce regular events.** It's hard to force yourself to update your site regularly. Even when you do, visitors have no way of knowing when there's something that makes a return visit worthwhile. Why not help everyone keep track of what's going on by promoting regular events (like a news section you update weekly or a promotional drawing that happens on a set date)?
- **Create feedback loops.** It's a law of the Web—good sites keep getting better, while bad sites magnify their mistakes. To help your site get on the right track, make sure there's a way for visitors to tell you what they like. Then, spend the bulk of your time strengthening what works and tossing out what doesn't.

Now that you have your website good-citizenship philosophy straight, it's time to learn how to build the ingredients every web community needs.

Website Community Tools

On its own, an ordinary website is a one-way medium, like cable television or newspapers. A web visitor surfs to your site, reads a few pages, and leaves. At that point, the conversation ends.

As websites become more community-oriented, they deepen and extend this interaction. The goal is to remind fickle viewers that you exist. If you're a big business, this approach is part of a branding strategy that establishes the value of your products in the mind of potential customers. If you're a small business, it could be part of the long courting process that leads up to a sale. And even if you're not selling or promoting anything, keeping visitors engaged with your website is the best way to boost traffic, reach new people, and remain the center of attention.

Websites differ in how deeply they focus on web community. Some sites add just a touch of community (for example, consider the product review system on an e-commerce site), while others go all the way with fan-run Facebook pages and discussion forums. Table 12-1 lists some of the ways a website can build a community. The rest of this chapter explains how you put them into practice in your own site.

Table 12-1. Different ways to reach your visitors.

Approach	Description	Level of Community	Learn More On
Email newsletters	Readers sign up, and you contact them whenever you want with the latest news. It's a good way to keep your community up to date, without forcing people to make a repeat visit.	Low. Newsletters are still a one-way conversation, and there's no opportunity for readers to respond.	Page 325
Blogs	You write regular, informal posts. Readers can keep up to date using a feed reader, and talk back by adding comments.	Medium. Blogs feel more conversational, and popular blogs attract piles of comments.	Chapter 13
Twitter	You send out brief snippets of text to the world with news, commentary, or random thoughts. Other people can pick up your theme and tweet (send out a message) to their fans.	High. At first, Twitter feeds look like a one-way conversation. But in reality, Twitter is part of a large, overlapping discussion that never ends.	Page 327
Groups	Readers join a forum where they can post messages and talk to each other. The only disadvantage is the discussion is limited to a pool of registered people.	High. Not only can you talk back to your visitors, they can talk to each other.	Page 329
Facebook	You create a fan page where you can post news, start discussions, and let other people chat about you. In some ways, a fan page is a hybrid of a newsletter and a group, with Facebook gluing it all together.	High. Like groups, but any one of the hundreds of millions of registered Facebookers can post.	Page 339

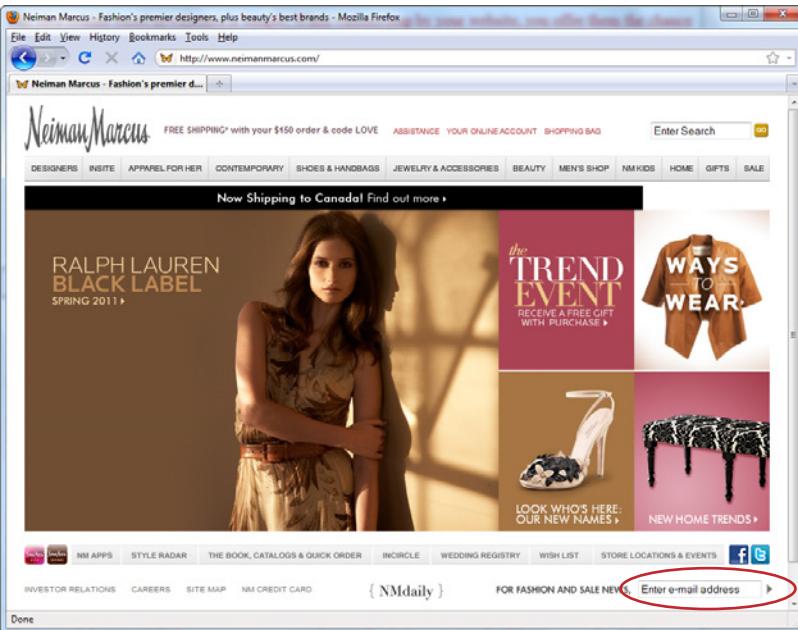
Email Newsletters

The idea behind email newsletters is simple. When visitors stop by your website, you offer them the chance to subscribe to your website's newsletter. With a little luck, visitors type in their email addresses, click a button, and give you a direct opening to their inboxes.

After that point, you send your visitors a periodic email with news or special offers. Usually, this email is no more frequent than once a month. After all, you don't want an exasperated recipient blocking your address and sending your messages straight to the junk mail bin.

Even in the modern web world of blogs, Facebook, and social media, email newsletters are still wildly popular. However, they introduce a few new challenges.

First, you need a small box somewhere on your website where people can type in an email address and sign up for your newsletter (see Figure 12-4). Creating this chunk of HTML is easy. Technically, it's called a *contact form*, and it uses a set of elements called *HTML forms* (page 453). The tricky part is what happens when someone clicks the sign-up button.



A screenshot of a Mozilla Firefox browser window displaying the Neiman Marcus website. The page features a large banner for Ralph Lauren Black Label Spring 2011, a "TREND EVENT" section with a woman in a dress, and a "WAYS TO WEAR" section with a jacket. At the bottom, there's a navigation bar with links like NM APPS, STYLE RADAR, and INVESTOR RELATIONS. A prominent red circle highlights a sign-up form in the bottom right corner. The form has a text input field labeled "FOR FASHION AND SALE NEWS Enter e-mail address" and a red arrow icon instead of a standard "Sign Up" or "Subscribe" button. To the right of the figure is a caption describing this design choice.

Figure 12-4:
Fashion retailers usually put a newsletter sign-up link on the front page of their sites. Here, a plain arrow icon takes the place of "Sign Up" or "Subscribe" button, which encourages spur-of-the-moment sign ups.

If you're a web coder working with a high-powered programming platform, contact forms are easy. When a visitor clicks a button, the browser sends her sign-up email address to a web application running on the web server, and that program stores the

address in a database. But mere mortals need a different approach. Essentially, you have two choices: borrow someone else's server-side script (a tiny scrap of code that runs on the web server) or use a form submission service.

The problem with the script approach is that you need one that works with your web hosting company's servers. The script code is different depending on the server-side programming technology they support. To solve this problem, you have three choices:

- **Ask your web host.** First, check with your web host to see if they have any ready-made scripts you can use to create a basic contact form. Typically, you'll use a small snippet of prewritten PHP or ASP code, which will extract the visitor's email address and send it to you in an email. You simply need to paste that code into your web page, and tweak a few details.
- **Search the Web.** If you can't find a ready-made script, check with your web host again—this time to find out if they support a server scripting language, and what that language is. Then you can use Google to hunt down a suitable script. For example, www.freicontactform.com/free.php has the barebones HTML and script you can use to collect email addresses on a web server that supports PHP.
- **Form submission services.** Finally, if your web host doesn't support server scripts, or you just don't want to wrestle with headaches a programmer would normally handle, you can use a free *form submission service*. Essentially, this service runs your form on its web servers, but emails *you* the data. To create your form, you simply choose the data you want to collect. The form submission company gives you a block of HTML that you plop into one of your web pages. You can find free form-creation services at www.emailmeform.com and www.123contactform.com. However, free services may put a tiny note at the bottom of the form (that says something like "this form powered by TheSuper-CoolFormCreationCompany"); if you don't want that, you need to pay.

Note: Contact forms are just the simplest example of data-collection forms. Using a form-submission service, you can create a form that collects a whole whack of information. For example, you might want to get a mailing address so that you can send out product samples. But be wary of asking for too much. The longer the form, the less chance that someone will fill it out and submit it.

Just because your website has a working sign-up box doesn't mean anyone will use it. People are justifiably paranoid of spammers and junk mail, and they only give up their email addresses if they think it benefits them. To get sign-ups, make the process easy—that means one box for an email address, and a button that says "Sign Up" or "Subscribe." You don't need names, phone numbers, birth dates, or any other information that might make a potential subscribe bail out. Then, sell the benefits of signing up with a simple, one-sentence description. These benefits could include hot deals, coupons, notification about special events, or opportunities that aren't available through the website.

Once you get guests' email addresses, use them. Web marketing research suggests that the average person looks at an email newsletter for just 51 seconds, so you need to include something obviously compelling that captures a reader's interest in that brief moment. Most newsletters are formatted in HTML, but it's best to compose them in an email program, so you don't attempt to use an HTML feature that some email clients might not support (like external or embedded style sheets). Use pictures sparingly and always add alternate text (page 184), because many email programs block images if they don't know who you are. Don't forget to include links that let readers jump from the email to the relevant page on your website, so they can follow up on breaking news or a hot new offer.

Twitter

Twitter is sometimes called a “micro-blogging” system. It lets anyone talk to the world by sending out text messages of 140 or fewer characters (hence the “micro” moniker). Of course, the success of your Twitter messages—called *tweets*—depends on whether anyone pays attention to what you say. Top twitterers have a huge crowd of followers that hang on every word. Often, followers can bring you more attention by retweeting your messages—that is, sending out your message to *their* followers.

You can follow a twitterer that you like in a number of ways. You could just watch their messages on an ordinary web page (Figure 12-5), but more serious twitterers keep up with a huge crowd of people using a third-party software tool or by having tweets delivered to their smartphone. (Similarly, twitterers can send tweets in a variety of ways, from typing them on Twitter's web page to using a smartphone app.)

Paradoxically, the source of Twitter's greatest strength is its greatest limitation: the 140-character limit that constrains twitter messages to a sentence or two. When it works, the limit forces twitterers to stick to concise, complete thoughts. For example, a single Twitter message might announce something new, link to a useful website, comment on current news, or reply to another tweet. The other great thing about Twitter is the breadth of people using it. You can follow tweets by everyone from the Dalai Lama to Lindsay Lohan. To see the most popular twitterers, visit <http://twitaholic.com/top100/followers>. To sign up for your own Twitter account, go to <http://twitter.com>.

Twitter can be a tricky medium. It takes time—perhaps a few weeks—to absorb the rhythm and flow of messages. The biggest mistake new twitterers make is to launch a feed, use it as a place to vent random or low-content thoughts (examples include “Obama sucks,” “I've lost my socks,” and “Am eating a piece of gristly chicken, right now!”), and then wonder why no one's paying attention. But if you tweet useful, insightful messages, you can gradually attract more attention to yourself, your brand, and your website.

The @ sign indicates a Twitter user name. It lets one twitterer mention or respond to another.

Martha's most recent tweet

A screenshot of a web browser displaying Martha Stewart's Twitter profile at <http://twitter.com/MarthaStewart>. The profile header shows a small photo of Martha Stewart and her name 'MarthaStewart'. Below the header, a tweet from her is displayed: 'http://ht.ly/3UVdE @thedailywag loves photo shoots!' with a timestamp of 12:44 PM Feb 11th via web. A red box highlights this tweet. Another red box highlights the URL 'http://ht.ly/3UVdE' in the tweet. To the right of the tweet, a sidebar provides account details: 'Verified Account', Name Martha Stewart, Location Katonah, NY, Web <http://www.themarthastewartshow.com>, Bio: curious inquisitive experiments entrepreneur who cares about the world we live in, 8,236 followers, 2,114,099 listed, and 2,215 tweets. Below the sidebar, there are sections for Favorites, Lists, Following (with a grid of profile icons), and an RSS feed for 'MarthaStewart's tweets'.

Tweets often point fans to interesting websites. But because there's so little space, twitterers use a service like TinyURL to shrink the site address first.

Figure 12-5:
Here is Martha Stewart's twitter stream, directly from the Twitter website.

Here are good guidelines for any new twitterer:

- **Take it slow.** Don't start twittering madly until you have a good feel for what works and what doesn't. To get a better feel, follow other twitterers. Use the Twitter search tool (<http://twitter.com>) to find tweets relevant to you (your business name, your area of expertise, your website topic), and see what other people are saying.
- **Share useful things.** Point out neat tools, recommend web links, and pass out snippets of advice. Don't use your Twitter account for relentless selling and self-promotion, because that turns everyone off.
- **Solicit opinions.** Ask others to interact with you. Here are examples of how top twittering companies get attention: offering advance copies of a book to review, asking about problems with current products, running a promotion where people chime in with stories or suggestions. The key is to involve the Twitter community *before* you have something to sell. They'll appreciate that, give good feedback, and maybe even help spread the word. But if you start twittering too late, everyone realizes you're not serious about the Twitter community.

- **Respond to other people.** Comment on someone else's tweet, or retweet something someone else said that overlaps with your site's subject area. This way, you're not just shouting from the rooftops. You're actually engaging with other people.
- **Show the human side.** It's not all about your cause, business, or professional aspirations. When people follow a twitterer, they expect to get a personal touch. So use Twitter to share comments that explain the real-life side of what you do.

Tip: There are two ways to achieve Twitter success. First, you can start off being such an important, wonderful, and famous person that no one wants to miss what you text from your mobile phone at 3:00 AM. But assuming you're not a celebrity, you need to take the second approach: Say something that's relevant to other twitterers. That way, others may notice your comments, follow up on them, and help bring your words to a wider audience.

Twitter mastery is outside the scope of this book. If you're just starting out, check out *The Twitter Book* (O'Reilly), which can help you think like a true twitterer, and provides great advice for Twitter-based promotion.

Groups

In the early days of the Internet, websites weren't at the heart of the action. Instead, the most interesting and lively interactions took place on a mammoth collection of online bulletin boards called *Usenet*. Sadly, Usenet fell into decline as the Web grew, spam became pervasive, and slick graphical sites became the norm. More recently, Google bought the collection of Usenet groups, which is experiencing a small renaissance as a part of Google Groups (see <http://groups.google.com>).

Before you create a group, you need to understand where it works and where it doesn't. Groups are *not* a good way to strike up a conversation with people who don't really know you and who don't have a vested interest in your website. Potential customers, curious web tourists, and people who stumble across your site aren't going to go through the trouble of joining a group to talk to you. You'll have a better chance luring them in with Facebook (page 339).

On the other hand, there are some situations where groups work better than loose-knit social media sites like Facebook. Here are some examples:

- **You have a niche topic.** This is a topic that attracts highly dedicated fans, but isn't already discussed somewhere else. This is tricky, because the Web already has *Glee* forums, forums for people to complain about bosses, vampire fan forums, and so on.
- **You provide product support.** If you're running a business, it makes sense to answer common questions in a forum so everyone can find the answers they need, rather than repeating yourself over and over again in email messages.

- Your group is an extension of the real world. In this case, the people in the group already know each other, and already have a reason to band together. For example, a group could be a convenient way for you to chat with the members in your knitting circle.
- Your group wants privacy. For example, if you run a support group for recovering addicts, you might want to shut out the Internet riffraff. In this case, you can create a Google group and limit its membership.

The best forums drive themselves. Once you get the right ingredients in place, a forum can succeed without you having to intervene. Think of forums as a dinner party that you host, and all you need to do is get the conversation started before making a polite retreat. And if you use forums to answer technical questions, you can reduce your workload immensely. For example, on many forums the emphasis is on customers or experts helping each other. That means forum members share information, advice, and answers, and you need to step in only to clear up a long-running debate.

Introducing Google Groups

The easiest way to create your own group is to use a free service like Google Groups. Here are some of its key details:

- When you create a group, you get a unique, easy-to-remember URL. That's the group address, and it never changes (unless you update it yourself).
- The group creator (you) controls who can and can't post. If the group gets busy, you can give other members some or all of your management powers.
- Google manages the registration process itself. That means you don't need to manually add and remove group members (although you can, if you like).
- Each group member can choose whether to read group messages online, or receive them in regular emails that Google sends automatically.
- Google's page layout is a frazzled web surfer's dream. It's easy to search posts, see all the replies to a post at a glance, bookmark the posts you want to follow, and more.
- Although Google displays ads in the corners of your group windows, it does its best to choose relevant ad content. For example, if the most common topic in a group is favorite DVDs, you're likely to see ads that promote Blu-ray DVD players and mail-order movie clubs.

You can learn more about Google Groups at <http://groups.google.com/groups/overview.html>.

Creating a group

To create a new Google Group, follow these steps:

1. Go to <http://groups.google.com>. Click the “Create a group” button to get started.

You need a Google account to create a group. If you already have one, sign in now. Otherwise, you need to register by clicking “Create an account.” Once you complete the process and activate your account, you’re ready to sign in and carry on.

Now, the “Create a group” page appears, as shown in Figure 12-6.

The screenshot shows the 'Create a group' page for Google Groups. At the top, there are two tabs: 'Set up group' (which is selected) and 'Add members'. Below these tabs, there are fields for 'Name your group' (containing 'Candy Collectors'), 'Create a group email address' (containing 'candy-collectors @googlegroups.com'), and a 'Group web address' (containing 'http://groups.google.com/group/candy-collectors'). There is also a 'Write a group description' field with the following text: 'A group of candy collecting enthusiasts comes together to discuss how to find the best candy, candy auctions, and appropriate strategies for preserving legendary candies for long periods of time.' Below this, it says 'Letters remaining: 105'. Under 'Choose an Access level', there are three radio button options: 'Public - Anyone can read the archives. Anyone can join, but only members can post messages and view the members list. Only managers can create pages and upload files.', 'Announcement-only - Anyone can read the archives. Anyone can join, but only managers can post messages, view the members list, create pages and upload files.', and 'Restricted - People must be invited to join the group. Only members can post messages, read the archives, view the members list, create pages and upload files. Your group and its archives do not appear in public Google search results or the directory.' At the bottom right of the form is a 'Create my group' button.

Figure 12-6:
Creating a Google group takes two steps. In the first, you define all the basic information about your group, including its name and email address.

2. Fill in all the information for your group.

The *group name* identifies your group, like *Candy Collectors*.

The *group email address* is a version of the name and it works as an email address or URL. You can’t use spaces, but dashes can do in a pinch; both *Candy-Collectors* and *CandyCollectors* work. The email address also becomes part of the group URL, so make sure it’s memorable.

The *group description* explains what the group’s all about, using two or three sentences.

The *access level* indicates who's allowed to post. If you want to create a completely open group that accepts all comers, choose *Public*, which makes sense for most web groups. Anyone who stumbles across your group can join at will, without your intervention. If you want to use the group solely as a place to post your own musings, choose *Announcements-only*. However, you're probably better off to put these announcements right on your website instead of in a group. Finally, if you want to micromanage who you let in, choose *Restricted*. That way, only the people you specifically invite can join the group.

Finally, you'll see a checkbox to allow adult content. If you leave this box unchecked, Google automatically blocks naughty posts, saving you some embarrassment.

3. Click the “Create my group” button.

Before continuing, you may have to copy some letters from a picture to prove you're a real person, not some sort of Google-group-generating program gone wild.

4. Choose whether to invite or add your initial group members.

Ordinarily, you invite new members to your group by email (see Figure 12-7). To actually become a member, each invitee needs to visit the site and opt in. But you can click “Add members directly” if you prefer to make these people automatic group members, with no acceptance required. Either way, recipients need to create a Google account (if they don't have one already) before they can post messages.

5. Fill in the initial set of group members.

Supply a list of email addresses separated by commas. When you finish creating the group, Google emails these people to tell them they can join the group (or, if you chose “Add members directly,” to tell them they already *are* members of the group).

Note: Don't worry if you don't have email addresses handy. You don't need to invite anyone now. You can return to this page to invite people later on, after you create the group.

6. If you chose to add members directly, set the subscription option.

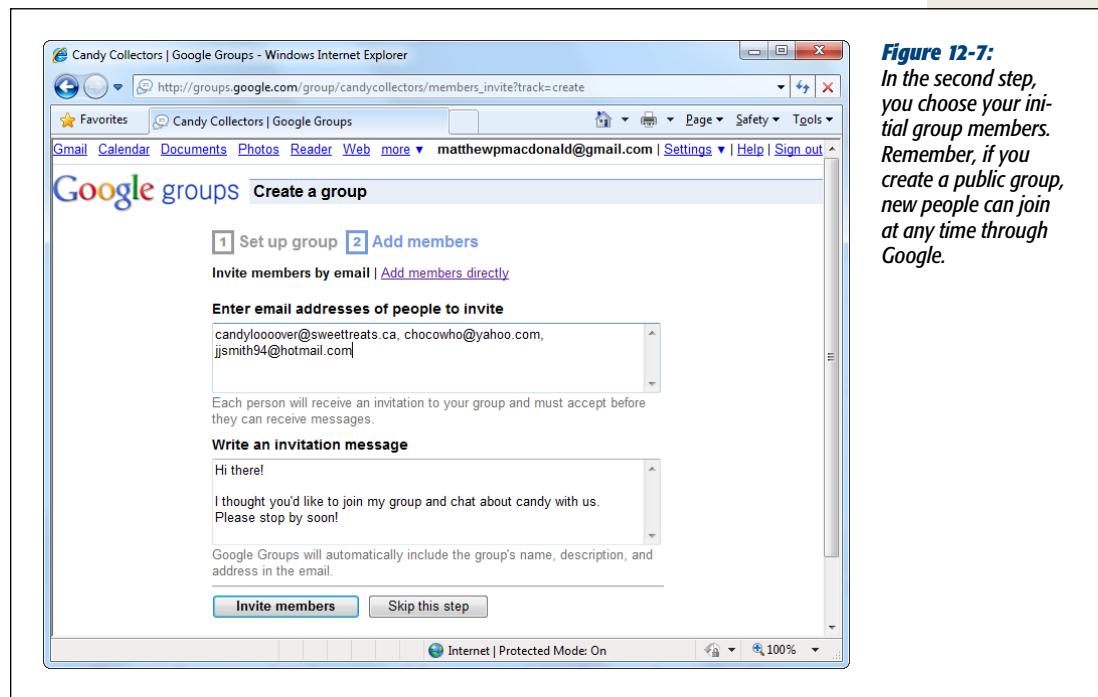
If you add rather than invite people, you see a section named “Email subscription options.” This lets you choose how these new members will interact with the group over email. (You don't get to make this choice if you're merely inviting new members. Instead, they'll pick their subscription option when they join the group.)

The subscription options include:

- **No email.** Group members have to read group posts in a browser, by visiting the group site.

- **Send email for each message.** Google sends every post to every group member. This is a bad idea unless it's a quiet discussion group.
- **One summary email a day.** Group members receive one email message per day (as long as there's at least one new post). This message contains a list of new message titles, with a link to the full text next to each one. This option is a handy way to stay on top of group activity, and keep an eye out for interesting posts.
- **One email with all activity.** Group members receive all the new content once per day in a single gargantuan email. This option won't clutter an email inbox as much as receiving each message separately, but in a busy group it results in long emails that make for impractical reading.

No matter which subscriber type you choose, group members can change this setting to match their personal preference later on.

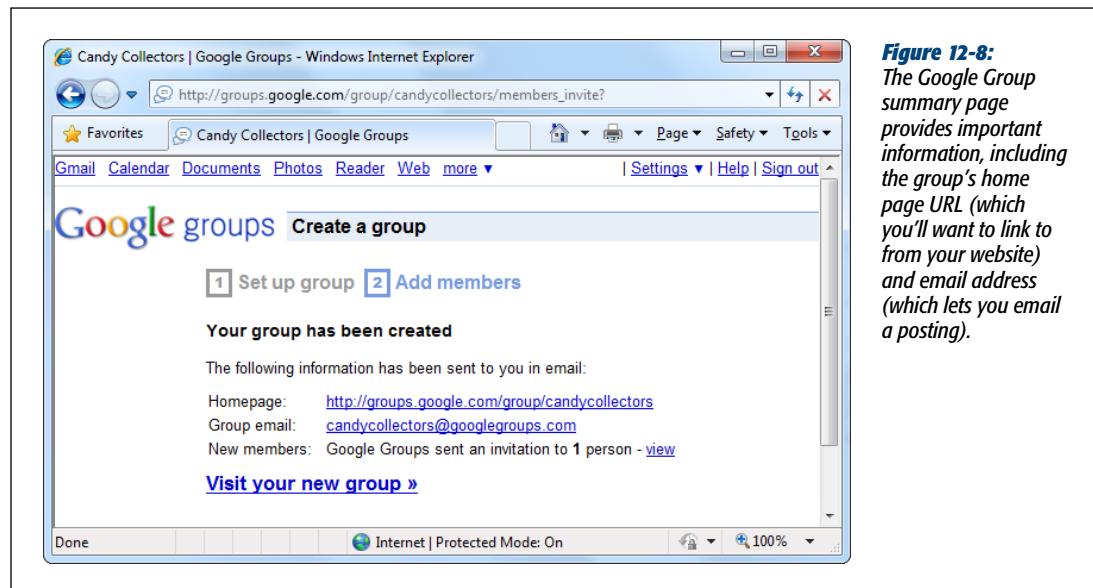


The screenshot shows a Windows Internet Explorer window titled "Candy Collectors | Google Groups - Windows Internet Explorer". The URL is "http://groups.google.com/group/candycollectors/members_invite?track=create". The page is titled "Google groups Create a group". It displays two steps: "1 Set up group" and "2 Add members". Under "Set up group", there is a link to "Invite members by email" and another to "Add members directly". A text input field contains email addresses: "candyloooover@sweettreats.ca, chocowho@yahoo.com, jsmith94@hotmail.com". Below the input field, a note says: "Each person will receive an invitation to your group and must accept before they can receive messages." Under "Add members", there is a text area titled "Write an invitation message" containing the text: "Hi there! I thought you'd like to join my group and chat about candy with us. Please stop by soon!". A note below says: "Google Groups will automatically include the group's name, description, and address in the email." At the bottom are two buttons: "Invite members" and "Skip this step". The status bar at the bottom of the browser window shows "Internet | Protected Mode: On" and "100%".

Figure 12-7:
In the second step, you choose your initial group members. Remember, if you create a public group, new people can join at any time through Google.

7. Enter a welcome message, and then click “Invite members” (or “Add members,” if you chose to add members directly).

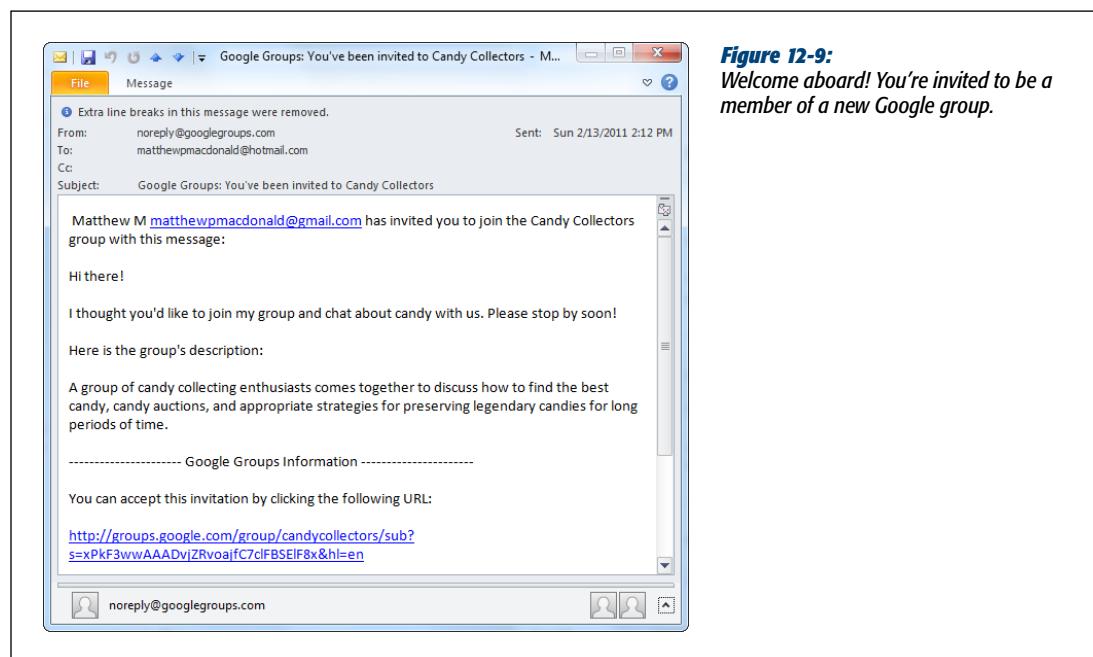
Google creates the group, and shows you a summary (Figure 12-8).



The screenshot shows a Windows Internet Explorer window displaying the Google Groups summary page for a group named "Candy Collectors". The URL in the address bar is http://groups.google.com/group/candycollectors/members_invite?. The page header includes the Google Groups logo and navigation links for Gmail, Calendar, Documents, Photos, Reader, Web, more, Settings, Help, and Sign out. Below the header, there are two main buttons: "Set up group" (highlighted with a blue box) and "Add members". A message states "Your group has been created". It provides the group's homepage (<http://groups.google.com/group/candycollectors>), group email (candycollectors@googlegroups.com), and a note about new members. A link "[Visit your new group »](#)" is present. At the bottom, the status bar shows "Done", "Internet | Protected Mode: On", and zoom level "100%".

Figure 12-8:
The Google Group summary page provides important information, including the group's home page URL (which you'll want to link to from your website) and email address (which lets you email a posting).

Sometime shortly thereafter, it sends welcome messages to the initial set of group members (see Figure 12-9).



The screenshot shows an Microsoft Outlook inbox window titled "Google Groups: You've been invited to Candy Collectors - M...". The message is from "noreply@googlegroups.com" to "matthewpmacdonald@hotmail.com" on "Sun 2/13/2011 2:12 PM". The subject is "Google Groups: You've been invited to Candy Collectors". The message body starts with "Matthew M matthewpmacdonald@gmail.com has invited you to join the Candy Collectors group with this message:". It continues with "Hi there!", "I thought you'd like to join my group and chat about candy with us. Please stop by soon!", "Here is the group's description:", and "A group of candy collecting enthusiasts comes together to discuss how to find the best candy, candy auctions, and appropriate strategies for preserving legendary candies for long periods of time.". Below this is a separator line "----- Google Groups Information -----". It then says "You can accept this invitation by clicking the following URL:" followed by a blue hyperlink: <http://groups.google.com/group/candycollectors/sub?s=PkF3wwAAADvjZRvoajfc7clFBSEf8x&hl=en>. The bottom of the window shows the recipient's email address "matthewpmacdonald@hotmail.com" and the sender's email address "noreply@googlegroups.com".

Figure 12-9:
Welcome aboard! You're invited to be a member of a new Google group.

Participating in a group

When you first head over to your group, you'll find that it's awfully barren. To get the discussion started, why not post the first topic?

Google gives you two ways to post a topic. You can add one right from the group's home page by clicking the "Start a new topic" link. Or, if you're really in a hurry, you can simply send an email message to the group email address. Google converts it into a group topic (Figure 12-10).

The figure consists of two vertically stacked screenshots. The top screenshot shows a Microsoft Outlook-style email client window titled 'Candy Sale - Message (HTML)'. The 'To' field contains 'candycollectors@googlegroups.com'. The subject is 'Candy Sale'. The message body reads: 'I just spotted discount candy on Amazon. Hurry and fill your virtual shopping carts!' and is signed off as 'Jake'. The bottom screenshot shows a Windows Internet Explorer browser window titled 'Candy Sale - Candy Collectors | Google Groups - Windows Internet Explorer'. It displays a single message from 'Jake Mashone' dated 'Feb 13, 2:21 pm' with the same content: 'I just spotted discount candy on Amazon. Hurry and fill your virtual shopping carts!'. Below the message are links for 'Reply', 'Reply to author', 'Forward', and 'Report spam'. On the right side of the browser window, there is a sidebar with options like 'Home', 'Discussions', 'Members', and a link to 'View this group in the new Google Groups'.

Figure 12-10:
Top: This email is about to be sent to a Google group.
Bottom: Once Google Groups receives the email, it becomes an ordinary posting (shown here). You can click "View profile" to learn more about the message poster or "More options" to reply, forward, or remove the message (assuming you're the group owner).

Of course, discussions are all about back-and-forth exchanges. Once someone posts a message, you can read it, and then click the Reply link to post a response. Google threads posts and replies, which means it groups them together so you can easily see what message goes with what topic (see Figure 12-11).

Note: If you scroll down to the bottom-right of the discussion page for your group, you'll find a small box of Google-supplied ads. Google chooses ads based on the content in your group, and refines its selection depending on which ads get the most clicks from group members.

The screenshot shows a Windows Internet Explorer window displaying a Google Group named 'Candy Collectors'. The main content area is titled 'Discussions' and lists three posts:

- White Chocolate - Oxymoron?**
It isn't chocolate. There is no cocoa. Case closed.
By Matthew M - 2:27pm - 2 messages - Report as spam
- Beware: Chocolate is Poisonous to Dogs**
Some find it a stunning fact, but chocolate can kill our doggie friends. White chocolate, not being chocolate, causes no problem.
By Matthew M - 2:26pm - 3 messages - Report as spam
- Candy Sale**
I just spotted discount candy on Amazon. Hurry and fill your virtual shopping carts!
Jake
By Jack Mahone - 2:21pm - 1 message - Report as spam

Below the posts, a message says 'No more topics in this group.' At the bottom left is a '+ New post' button. On the right side of the page, there are several sidebar links: Home, Discussions (+new post), Members, About this group, Edit my membership, Group settings, Management tasks, Invite members, View this group in the new Google Groups, and Group info (Language: English, Group categories: Not categorized, add a category). The status bar at the bottom indicates 'Internet | Protected Mode: On'.

Figure 12-11:
When browsing a group, you see a list of topic posts only, not replies. However, each entry clearly indicates the number of replies. In this example, there are three posts and a total of six messages. The first post has one reply, and the second post has two replies. (Click the post title to see the full post and any replies.)

You now have a fully functioning group. From this point on, the challenge isn't in using the group, it's in attracting enough interesting people so that it becomes a lively community.

Managing your group

When ordinary members visit their group, they have the option to post new messages, reply to existing ones, or change their delivery settings (by clicking the "Edit my membership" link). Group members can use this last option to have group messages automatically emailed to them or to see a summary of group activity.

On the other hand, when the group *creator* visits the group, additional links appear. On your group page, you see an "Invite members" link, which lets you send welcome messages to a new batch of groupies. The "Group settings" and "Management tasks" links let you take control of a lot more.

The "Group settings" section is chock full of options, organized into a few subgroups. Here's a quick rundown of what they offer:

- **General.** This page shows basic group information, like your group name, description, URL, and email address. Click the Edit button to change these details.

- **Access.** These settings let you define who can perform various tasks. For example, you can control who can read and create posts (anybody, or only group members), who's allowed to invite new members (just you, or any group member), and who's allowed to join. This last option is the most interesting. You can allow everyone, restrict the group to just people you invite, or force strangers to apply for group membership. If you use the last choice, anyone can apply to join, but you have the chance to review the application and give the final vote of acceptance or refusal. You can even tell Google to give hopeful applicants a specific question. You can then review their answers to determine whether they're group-worthy. Lastly, you can use the access settings to turn on *moderated messages*. With moderated messages, Google sends every new message to you before it posts it. Messages won't appear until you give them a thumbs-up (and if you don't, they never reach the group). Use moderated messages only if you have a lot of spare time.

FREQUENTLY ASKED QUESTION

Group Restrictions

Should I restrict people from joining my group or posting messages?

It's tempting to force group members to apply to your group, but resist the ego trip. On the Web, people are impatient and easily distracted. If you place barriers in the way of potential group members, they may just walk away.

On the other hand, there are some cases where restricted group membership makes a lot of sense. Two examples are when you want to discuss semi-secret information, like company strategies, or if you're afraid your topic might attract the wrong kind of crowd. For example, if you set up a group called Software-Piracy to discuss the social implications of software piracy, you might find yourself deluged with requests for the latest versions of stolen software. As a general rule, restrictions make sense only if you use them to maintain the quality control of your group.

The same holds true for message moderation. Most healthy online communities are self-regulating. If a member inadvertently offends the general community, others will correct him or her; if it's deliberate, most will eventually ignore the provocation. You might need to step in occasionally to ban a member, but screening every message is overkill. It also adds a huge amount of extra work for you, and severely cramps the dynamic of your group, because a new message won't appear until you have the chance to review it, which will usually be several hours after the poster wrote it. For fans of the Web who expect instant gratification, this isn't good news.

You'll find the settings for restricting people and moderating messages in the Access section of the "Group settings" page.

- **Appearance.** If you want your group to stand out from the crowd, you can use these options to give your group site different fonts and a snazzy color scheme.
- **Navigation.** This section lets you hide some of the links shown on the main group page. For example, if you don't want to let group members upload and share files, you can remove the Files link.
- **Email delivery.** If some of your group members receive messages by email, you can use these settings to tweak the footer text and control what happens if a member replies to an emailed posting. Ordinarily, the reply is posted alongside

the original message for the whole group to see, but you can direct replies to just the original poster or to the group owner (you).

- **Categories.** Here, you can define a category for your group (like People→Relationships or Health→Addictions). Once you do so, Google searchers can more easily stumble across your group.
- **Advanced.** This section lets you perform some low-level administrative tasks, like setting up a group that's stored on another server, or deleting your group altogether.
- **Spam.** This section lets you control how Google deals with suspected junk messages. For example, it can delete them immediately or put them in the moderated messages section (which is the default).

While the “Group tasks” link presents you with a wide range of settings, the “Management tasks” link focuses on a single concept—dealing with group members. Using the “Management tasks” page, you can review the full list of group members, see who hasn’t responded to a group invitation, ban troublemaking posters, and give other members managerial powers (see Figure 12-12).

The screenshot shows a Mozilla Firefox browser window with the URL http://groups.google.com/group/candy-collectors/manage_members. The main content area is titled "Management tasks" and lists "All members (5)". A dropdown menu for "Set membership type" is open over the first row, showing options: "Email", "Regular member", "Manager", and "Unsubscribe". The "Manager" option is highlighted. The table below shows five members with their details: Sarah (selected), jon@patrakycouk, mohammad_1230@yahoo.com, matthewpmacdonald@gmail.com - me, and zaphod.b@42answer.ca. The columns include Name, Joined, Delivery, Posting, and Edit links. To the right of the table is a sidebar with links: Home, Discussions, Members, Pages, Files, About this group, Edit my membership, Group settings, Management tasks, and Invite members. A callout bubble points to the "Manager" dropdown with the text "Sarah is selected (note the checkmark by her name on the left)".

Figure 12-12:
Here, the group member Sarah is selected (note the checkmark by her name on the left). Using the drop-down lists at the top, you can quickly change her message delivery options or assign her a different access level. In this example, she’s about to be made into a manager, giving her the ability to remove posts, invite new members, and change group settings (but not delete the entire group). Other options include Unsubscribe, which would remove her from the group, and Ban, which would remove her with extreme prejudice, so she would never be allowed to rejoin.

Facebook

It can be tough to build a group from scratch. If a new group doesn't catch on quickly, it becomes a lonely place where no one wants to linger. That's why an increasing number of web dwellers don't try to do it alone. Instead, they bring their audience to an existing community—one that's set up around a social networking site. And when it comes to social networking, no company is better known than Facebook.

Facebook began as a way for college students to keep in touch with each other. In only a few years, it mushroomed into a social site where hundreds of millions of ordinary people track down everyone from long-lost loves to faintly remembered high-school acquaintances. Thanks to fan pages (the feature you'll learn about in this section), Facebook has even become a tool for businesses and nonprofit organizations—one with unique advantages and limitations.

Creating a fan page

A *fan page* is a Facebook page that sets out to promote just about anything, including companies, causes, brands, books, television shows, and more. If you're trying to promote yourself, a fan page is a good choice. It's a great tool for musicians, comedians, journalists, and just about anyone who wants to talk to a broader audience. For example, www.facebook.com/kristof is the fan page for *New York Times* columnist Nicholas D. Kristof. He uses it to comment on current affairs and discuss the issues of the day with readers. On an entirely different but more delicious note, www.facebook.com/benjerry is a Facebook fan page for Ben & Jerry's ice cream. They use as a chattier, less formal version of their website, complete with whimsical discussions about now-abandoned ice cream flavors.

A fan page is similar to a personal Facebook page, but it's better suited to promotion. That's because anyone can visit it and read the content on a fan page, even if they don't have Facebook accounts of their own. Those who do have accounts can do the usual Facebook things—click Like to follow the fan page, post on its wall, and join in any of its discussions.

Note: A personal Facebook page (known as a *profile page*) is more restrictive than a fan page. It's better suited for talking to your friends or networking with business contacts. But a fan page is a better way to promote yourself, your business, or your cause to the masses of people you don't know.

Here's how to create a Facebook fan page:

1. To create a fan page, go to www.facebook.com/pages/create.php.

Facebook splits this page into two sections (see Figure 12-13). On the left, you can create a community page, which is a shared page that, if popular enough, the Facebook community will maintain. On the right, you can create an "official" page, which is what you want.

If you don't have a Facebook account, don't worry—you can create one on the way.

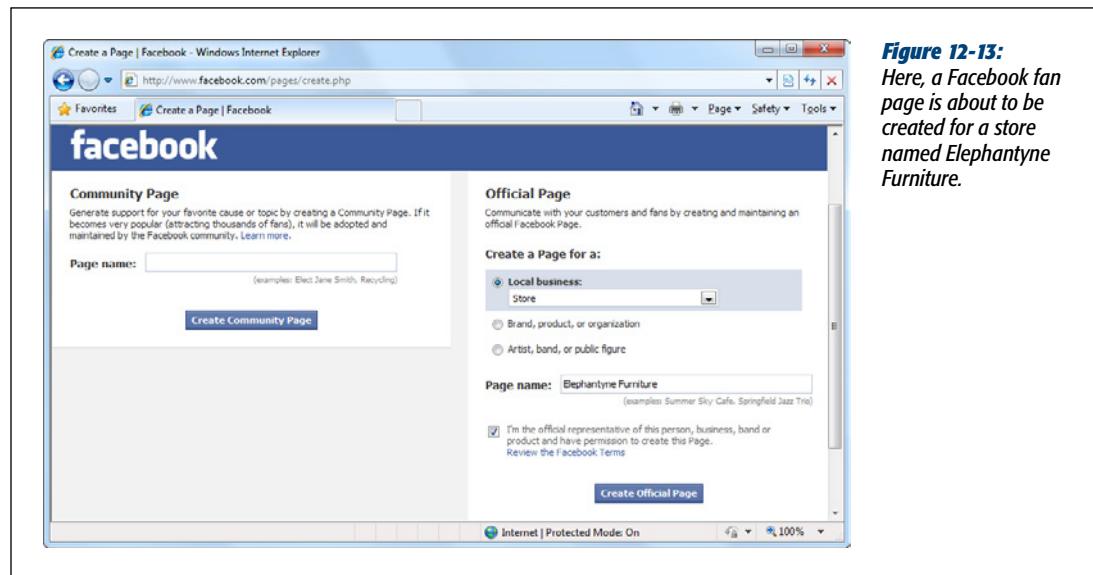


Figure 12-13:
Here, a Facebook fan page is about to be created for a store named Elephantyne Furniture.

2. Choose who you're creating the page for.

Your choices are “Local business,” “Brand, product, or organization,” or “Artist, band, or public figure.” If these options seem to combine different types of pages into one category, don’t worry. Once you choose an option, a more detailed list appears underneath that lets you specify exactly what type of business, brand, organization, or person you really are (the fan page in Figure 12-13 chooses “Store” from that list).

3. Enter a page name.

Good examples include “Larry S. Tindleman” or “Larry’s Polka Band” or “Tindleman World-o-Shoes,” depending on what you want to promote. The page name can include spaces. The name appears prominently on the fan page, and Facebook uses it in the fan page URL.

4. Click the “I’m the official representative box” if you are.

In some cases, Facebookers have started fan pages for brands they love. Often, the company that owns the brand later takes over the page, but sometimes it lets the fan-driven page continue.

5. Click Create Official Page.

You’ll now need to either sign in with your Facebook account or create a new one.

To start a new Facebook account, you need to supply just a few details: your email address, a password you’d like to use, and your birth date. To confirm your account, click on a link in a message that Facebook emails you.

When you finish signing in or registering, Facebook creates your fan page.

Every fan page starts out with two publicly viewable tabs: Wall (where you and your fans can post short messages) and Info (where you can share basic information about yourself or your business). Your fan page also includes a Get Started tab that's for your eyes only. Facebook starts you there (Figure 12-14).

The screenshot shows a Windows Internet Explorer window displaying the Facebook Get Started page for a fan page named "Elephantyne Furniture". The page is titled "Elephantyne Furniture > Get Started". It features four numbered steps: 1. Add an image (with a large question mark placeholder), 2. Tell your fans (with an Import Contacts button), 3. Post status updates (with a Post Update button), and 4. Promote this Page on your website (with an Add Like Box button). On the left sidebar, the "Get Started" tab is highlighted. A red callout box points to the "Get Started" tab with the text "Your fan page starts with just three tabs. Click to switch from one tab to another." Another red callout box points to the "Edit Info" link in the top right with the text "Click here to edit everything in your fan page profile."

Figure 12-14:
Right now the Elephantyne Furniture fan page is a blank slate, with no picture or information. The Get Started page provides quick links to some of the most common setup tasks, like uploading a profile picture, importing a list of email contacts, posting a status update, and generating the HTML for a Like box, which you can put on a page in your website. ("Liking" is how Facebook fans reward the people and things they like. The more people that click Like for your website, the better your Facebook reputation.)

6. Click the Edit Info link at the top of the page, just under the page title.

Now it's time to supply basic details about your page, along with a picture. You'll see a new, multitabbed page where you can supply piles of information about your new fan page. You choose a category in the sidebar on the left, and fill in the corresponding information on the right.

To start, choose Basic Information. You can add an address and description (which is important, because it appears on the Info tab of your fan page). Click Save Settings when you finish.

Next, click Profile Picture. Here, you can upload a picture that will sit in the top-right corner of your fan page. This detail is critically important, because it's the first thing viewers see, and it establishes your identity.

7. To return to your fan page, click the View Page button (in the top-right corner).

Although you might not feel like you've done very much, your Facebook fan page is now live and accessible to anyone on the social network. People outside of Facebook can see it too, but it will take a while before it turns up in Google's search engine. In the meantime, you can link to it directly using the fan page URL.

8. Take note of the URL for your Facebook fan page.

The quickest way to get your page's URL is to look at the web browser's address bar. Your URL is everything up until the question mark. It includes the Facebook site, page name, and a series of numbers.

For example if you see this in the address bar:

`http://www.facebook.com/pages/Elephantyne-Furniture/129277497141285?sk=wall#!/pages/Elephantyne-Furniture/129277497141285`

Your fan page URL is this:

`http://www.facebook.com/pages/Elephantyne-Furniture/129277497141285`

Tip: Fan page URLs are messy, thanks to the long series of numbers at the end. But you've probably noticed that big players get better URLs, with no numbers at the end. You, too, can apply for a number-free Facebook vanity URL, but first you need at least 25 fans. (Fans are people who have clicked the Like button on your page.) For more information, see <http://tinyurl.com/4z9wzlz>.

GEM IN THE ROUGH

Becoming Your Fan Page

In the past, fan page creators have had to use their personal Facebook account for a number of tasks. This isn't always perfect. For example, it's a bit awkward if you want to keep some privacy between you and your business, or if you don't want to take on the role as official spokesperson for your organization (perhaps the fan page really belongs to a whole team of people, or you don't want to challenge the brand by introducing your personal presence).

But recently, the makers of Facebook introduced a new feature that lets you do many of the things you would normally do with your personal Facebook account—network with your friends, connect with other fan pages you like, and so on—using your fan page account.

To switch to your fan page account, click the Account button in the top-right corner and choose Use Facebook As Page. Facebook shows a list of all your fan pages. Click the

Switch button next to the fan page you want to become.

Once you assume the identity of your page, you can leave messages on other people's walls, track the activity on your fan page, add comments to ongoing discussions, and "Like" other businesses (which gives you a great avenue for cross-promotion). For example, Elephantine Furniture might choose to "Like" a wholesaler, marketer, or supplier it works with. On Facebook, it will clearly indicate that "Elephantine Furniture" likes the person or business, without revealing the personal details of the person that manages the fan page.

When you're ready to switch back to your personal profile, click the Account button again and choose the "Switch back" command. The exact text depends on your name—for example, if you're registered with Facebook as Rob Sampson, you'll see a "Switch back to Rob" command.

Adding tabs to your fan page

Your fan page starts out with just two public tabs: Wall and Info.

The Wall is a central posting place for you and your fans. People read your wall to take the pulse of your page. If it's full of lively chatter, it's a good sign that your page is thriving. But the most important part of the wall is the way it delivers your messages to fans. You post a status update, photo, website link, or video, and your fans get notified immediately (Figure 12-15).

Tip: Remember, Facebook is all about community. You don't need to reply to every comment, but you should talk back often. Offer thanks when praised and apologies when criticized. See <http://tinyurl.com/6amf47k> for the Facebook fan page of a small jewelry seller that gets it right.

The Info tab is just as essential, but not nearly as exciting. It shows the details you set in the Basic Information section of your profile, such as your description.

This is a reasonable start, but a respectable fan page needs more tabs. Technically, Facebook calls these tabs *apps* (which is shorthand for *application*). Here are some of the apps that Facebook fan pages often include:

facebook Search

Elephantyne Furniture 

Local Business

Wall Everyone · Ele

Elephantyne Furniture
Elephantine Furniture is having a Winter Sale!
36 minutes ago

RECENT ACTIVITY

Elephantyne Furniture joined Facebook.
Elephantyne Furniture changed their Description.

0 people like this

Figure 12-15:
It all starts when
someone clicks Like
on your page (top).
Now, when you post
a message from your
Wall (middle), the
information travels to
your fan's news feed
(bottom).

facebook Search

Elephantyne Furniture 

Local Business 

Wall Everyone · Elephantyne Furniture

Share:  Status  Photo  Link  Video

Announcing a safety recall on all tusk utensils

Everyone Share

0 people like this

facebook Search

News Feed Top News · Most Recent

Share:  Status  Photo  Link  Video

What's on your mind?

Elephantyne Furniture
Announcing a safety recall on all tusk utensils
3 hours ago · Like · Comment

Elephantyne Furniture
Elephantine Furniture is having a Winter Sale!
4 hours ago · Like · Comment

There are no more posts to show right now. 

- **Events.** The perfect way to advertise an upcoming function, whether it's a concert in a local pub or a one-day sale on mismatched socks.
- **Photos.** Facebookers expect to be able to see you, and photos make that happen. Often, Facebook pictures are informal and focus on people (for example, a business might upload pictures of its employees or a social event). It's up to you whether you want to be the sole picture uploader or let your fans in on the action.
- **Notes.** A note is, in Facebook-speak, a short posting of the sort you might see on a blog (see Chapter 13). But the truly neat thing about notes is the way they can travel to your fans and then to other interested people in a way that's similar to but more powerful than wall messages. That's because Facebook gives members the additional ability to share notes (essentially, to repost them), as shown in Figure 12-16.

The screenshot shows a Facebook News Feed in Mozilla Firefox. On the left, the sidebar includes 'Notes' and 'Links'. The main feed displays a post from 'Lisa Chang' titled 'What Inspires Us' with a note about elephants. Below it is a comment from 'Rakesh Sureet'. A red arrow points from the text 'A post from the Notes section of Elephantyne Furniture.' to the original post. Another red arrow points from the text 'Lisa Chang clicks Share to post it to her wall, where others can see it.' to the 'Share' button in the post's options menu. The status bar at the bottom says 'Stopped'.

Figure 12-16:
Lisa Chang is a fan. When she sees a wall post from Elephantyne Furniture, she can Like it or comment on it. But when she sees a Notes entry, she has the additional option of sharing it on her wall by clicking Share. Now, her friends can see it and comment on it, deepening the reach of your fan page.

A post from the Notes section of Elephantyne Furniture.

Lisa Chang clicks Share to post it to her wall, where others can see it.

- **Discussion Boards.** Wall messages are a disorganized free-for-all. Discussions, on the other hand, group a topic's postings and replies in a single, cohesive thread. You can start your own discussion (for example, you can ask for suggestions or feedback), or you can let other Facebookers chat it up.
- **Reviews.** If you can get people to express their feelings about you or your business, you can build their loyalty and get useful feedback.
- **Static FBML.** Use FBML (short for Facebook Markup Language) to create your own tab. FBML includes the standard HTML and CSS that you already know about, so you can copy markup straight from a web design tool (like Dreamweaver or Expression Web) into a Static FBML tab. Fan page creators often use this feature to create a richly formatted welcome page.

Along with these standard Facebook apps, you'll find plenty of third-party tools that you can incorporate in your pages. Search around (type "Facebook apps" into a search engine), and you'll find programs that run polls, play games, and more.

Tip: At some point, you may decide that you want visitors to start out at a tab other than the wall when they visit your fan page. This is often the case if you add the Static FBML app and use it to create a richly formatted welcome page. To change the initial tab, edit your profile, click the Manage Permissions tab, and choose the tab you want from the Default Landing Tab list.

There are two ways to add a tab to your page. The best starting point is to follow these steps:

1. Edit your profile (look for the Edit Info link).
2. Click the Apps tab.

This lists apps that you technically have, but aren't displaying in your fan page. It also lists a section of apps you might like.

3. Click the Edit Settings link next to the app you want.

A pop-up window appears.

4. To add this app to your Facebook fan page, click Add, and then click Okay.

This works for most of the apps mentioned above, but not all. (For example, you won't find the Reviews app here.) If you follow these steps but you don't see the app you want, move on to this approach:

1. Type the name of the app you want in the search box at the top of the Facebook page.

The site responds with a list of all the Facebook items that match your search so far, including apps.

2. When you see the app you want appear in the list, click it.

Facebook takes you to the page for that app.

3. Click the “Add to my page” link.

You’ll find the app added to the sidebar on the left.

Be warned that this approach doesn’t work for the special Events and Notes apps, because they’re always attached to your fan page (even though they aren’t necessarily visible). To switch them on, use the first set of steps.

Promoting your Facebook page on your website

Now that you’ve crafted the perfect fan page, it’s time to use it on your website. You have several promotional options, including:

- **Facebook badge.** This is a small box that advertises your Facebook fan page on a web page. When someone clicks it, they move from your website to your Facebook fan page. To create a badge, go to www.facebook.com/badges and click Profile Badge. Optionally, click the “Edit this badge” link to customize the look of your badge. Then, click the Other button and get the HTML, which you can copy into any of your web pages.
- **Like button.** This is a Facebook-styled Like button, which you can slap on any page. If a Facebooker visits your page and clicks this button, it’s the same as clicking Like on your actual fan page: It establishes a relationship that boosts your ranking, and allows information to flow from your page to your fan’s Facebook page. To get a Like button, go to <http://developers.facebook.com/docs/plugins> and click Like Button. Fill in your Facebook fan page URL, tweak the other options if you want to change the button’s appearance, and then click Get Code.
- **Like box.** This is a panel that includes a summary of what’s happening with your fan page—for example, recent wall and note postings. It also includes the ever-important Like button. To get a Like box, go to <http://developers.facebook.com/docs/plugins> and click Like Box. Fill in your Facebook fan page URL, tweak the other options if you want to change what the Like box looks like, and then click Get Code (Figure 12-17).

Typically, you’ll position a Like box or Facebook badge in a separate column on your fan page, using the CSS positioning properties you learned about in Chapter 9.

Tip: To pick up basic Facebook skills and hone your promotional strategies, check out *Facebook: The Missing Manual*.

The screenshot shows the 'Like Box - Facebook Developers' page in Windows Internet Explorer. On the left, a sidebar contains configuration options: 'Facebook Page URL' (set to 'http://www.Elephantyne-Furniture.com'), 'Width' (set to 292), 'Color Scheme' (set to 'light'), 'Show Faces' (checked), 'Stream' (checked), and 'Header' (checked). Below these is a 'Get Code' button. The main area on the right displays a preview of the Like box. It features a 'Find us on Facebook' bar at the top with the page's logo and a 'Like' button. Below this is a section titled 'Elephantyne Furniture' with the heading 'What Inspires Us'. It lists three posts: 'Announcing a safety recall on all tusk utensils' (posted 5 hours ago), 'Elephantyne Furniture Elephantine Furniture is having a Winter Sale!' (posted 6 hours ago), and a comment '1 person likes Elephantyne Furniture.' with a small profile picture of a person named Lisa. At the bottom of the preview is a 'Facebook social plugin' button.

Figure 12-17:
To create a Like box, you fill in the settings on the left and look at the preview on the right. This Like box includes all the frills: Show Faces (to show the faces of some of your followers), Show Stream (to show recent news content), and Show Header (which adds the "Find us on Facebook" bar at the top).

Blogs

A traditional website is the gold standard of the web world. It's infinitely flexible—able to chronicle a personal trip to Machu Picchu just as well as it powers an e-commerce storefront.

However, there's something distinctly unspontaneous about a website. For example, imagine you want to post a piece of gossip about a celebrity sighting in your hometown. Before you can share your thoughts with the rest of the word, you need to pick a web page file name, decide what HTML markup you'll use, determine how you'll link your page to other pages (and edit them if necessary), and so on. None of these tasks is really that difficult, but taken together, they're enough to discourage casual web authors from writing anything that doesn't seem worth the trouble.

That's where blogs fit into the picture. Blogs are a self-publishing format that gets your thoughts online quickly and easily, while avoiding the headaches of website management. They're a fresh, straightforward, and slightly chaotic way to communicate on the Web. To maintain a blog, you publish short entries whenever the impulse hits you. High-powered blogging software collects, chronologically organizes, and presents your blog posts on web pages. That means that if you don't want to fuss with the fine details of website management, you don't need to. All you need to worry about is sending in postings—and with some blogging software, that's as easy as firing off an email.

In this chapter you'll learn how blogs work, and you'll see how to create your own blog with Blogger, one of the Web's leading free blogging services.

Understanding Blogs

The word “blog” is an abbreviation of *web log*, which makes sense because blogs are logs of a sort—regular, dated blurbs, like a cross between a diary entry and a posting in a discussion forum. Blog is also a verb, as in “I just ate at a terrible restaurant; when I get home I’m going to blog about it.” Figure 13-1 dissects the anatomy of a basic blog.

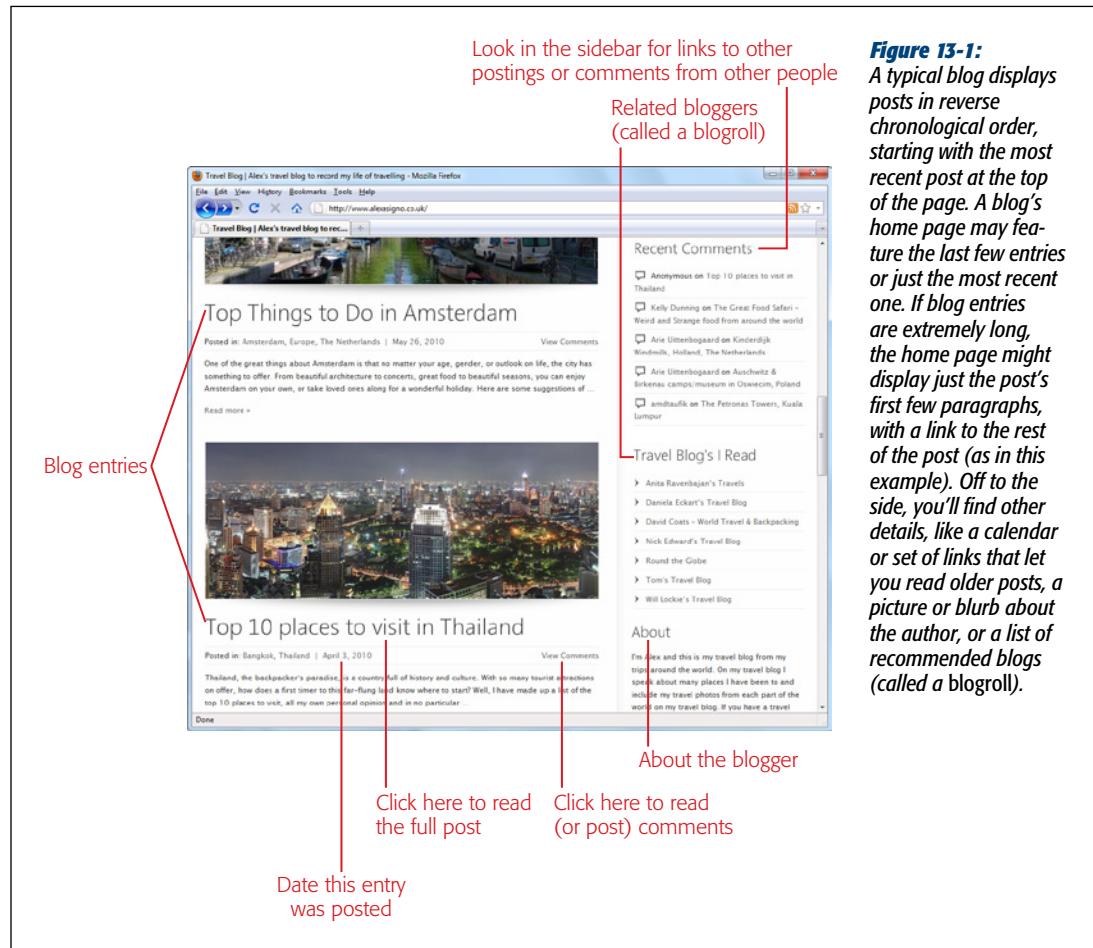


Figure 13-1:
A typical blog displays posts in reverse chronological order, starting with the most recent post at the top of the page. A blog’s home page may feature the last few entries or just the most recent one. If blog entries are extremely long, the home page might display just the post’s first few paragraphs, with a link to the rest of the post (as in this example). Off to the side, you’ll find other details, like a calendar or set of links that let you read older posts, a picture or blurb about the author, or a list of recommended blogs (called a blogroll).

Although blogs simplify web postings, it’s unfair to say that they’re just a simplified way to work the Web. Rather, blogs are a wholly different form of online communication. And although there’s no definitive test to determine what is or isn’t a blog, most blogs share several characteristics:

- **Blogs are often personal.** You can find topic-based blogs, work-based blogs, political blogs, and great numbers of blogs filled with random, offbeat musings. No matter what their mission, however, blogs usually emphasize the author’s

point of view. They rarely attempt to be objective—instead, they’re unapologetically idiosyncratic *opinions*.

- **Blogs are organized chronologically.** When you design a website, you spend a lot of time deciding how best to organize your material, often using menus or links to guide visitors through an assortment of topics. Blogs take a radically different approach. They have no organization other than ordering your postings chronologically. Anything else would just slow down restless bloggers.
- **Blogs are updated regularly.** Blogs emphasize fast, freewheeling communication rather than painstakingly crafted web pages. Bloggers are known to add content obsessively, often several times a week. Because blog entries are dated, it’s glaringly obvious if you don’t keep your blog up to date. If you can’t commit to blogging regularly, don’t start a blog—set up a simple web page instead.
- **Blogs are flexible.** There’s a bit of blog wisdom that says no thought’s too small for a blog. And it’s true—whether you write a detailed discussion on the viability of peanut-butter Oreos or a three-sentence summary of an uneventful day, a blog post works equally well.
- **Blogs create a broader conversation.** Blogs form communities more readily than websites do. Not only are blogs more conversational in nature, they also support comments and links that can tie different blogs together in a conversation. If someone posts an interesting item on a blog, a legion of fellow bloggers follows up with replies, blog postings, Twitter messages, and Facebook Likes. Scandalous blog gossip can rocket around the globe in a heartbeat.

Blogs occupy a specialized web niche, distinct from a lot of the other types of sites you’ve seen. For example, you can’t effectively sell a line of trench coats for dogs on a blog. But many people start blogs *in addition* to ordinary websites. This is a great combination. Visitors love blogs because they crave a glimpse behind the scenes. They’re also sure to visit again and again if they can count on a regularly updated blog that offers a steady stream of news, gossip, and insight.

Note: A significant number of big businesses have found that they can make their companies seem friendlier, more open, and more accessible by adding a blog with regular postings from a high-placed manager, programmer, designer—or even the CEO.

Popular Blogs

The actual content of a blog isn’t fixed; it can range widely, from political commentary to personal travelogues. The best way to get a feel for the blogosphere is to check out some popular examples.

One great starting point is Technorati (<http://technorati.com>), a search engine that indexes blogs. Head there, and you’ll find lists of the most popular blogs, and updates that show which blogs are on the uptrend (or downtrend).

However, it's important to realize that blogs are a *long tail* technology—in other words, there's a pile of super-popular blogs, but far, far more blogs that play to a small- or medium-size audience. To get a real feel for what blogs provide, go beyond the heavy hitters on Technorati (which is dominated by media, news, and political blogs), and look at some smaller gems. Here's a random selection of just a few:

- <http://stefaniewildertaylor.com>. Mommy blogs—blogs where a parent details the trials and tribulations of raising kids—are wildly popular. They allow bloggers to exchange tips (and share the pain) with a huge virtual community of parents. The range of topics is huge, from parents dealing with the serious medical problems of their children to the lighthearted comedy of potty stories. Stefanie Wilder-Taylor, a one-time comedian and writer, stands out with a blog that's profanely funny *and* serious (among other topics, her blog chronicles the challenges of mothers battling alcoholism).
- www.rotten gods.com. Since their inception, blogs have provided a way for people in the midst of war, famine, and civil strife to tell their stories. On his Rotten Gods blog, Fariborz Shamshiri posts his thoughts on life and religion in Iran. English readers flocked to his blog for up-to-the-minute information about the contested 2009 Iranian election.
- www.schneier.com. As you probably expect, blogs are a great place for computer geeks of all stripes. Computer security whiz Bruce Schneier analyzes all manner of security issues, from credit-card skimming to stolen elections and terrorist attacks. And if you're looking a less-credentialed computer nerd to follow, visit the curiously popular blog by Wil Wheaton (<http://wilwheaton.typepad.com>), the actor who played the nerdy upstart Wesley Crusher on *Star Trek*.
- <http://thesartorialist.blogspot.com>. Scott Schuman single-handedly created the fashion blog format while on paternity leave with his infant daughter. Out on the streets of New York, he took pictures of people and fashions that caught his eye. His blog, which includes his pictures and occasional comments, skyrocketed in popularity. The resulting fame transformed him into a photographer, author, and sought-after fashion legend—which is surely a testament to the power of blogging.

The list goes on—from journalists to hobbyists to sports heroes to porn stars, it seems that almost everyone's willing to psychoanalyze their life or chat about water-cooler topics with an audience of millions via a blog.

WORD TO THE WISE

The Hazards of Blogging

There's something about the first-person nature of a blog that sometimes lures people into revealing much more information than they should. Thanks to reckless moments of blogging, lovers have discovered their cheating spouses, grandmothers have read memorable accounts of their granddaughter's sexual conquests, and well-meaning employees have lost their jobs.

The dangers of impulse blogging are particularly great in the working world. In most countries, companies have the ability to fire employees who make damaging claims about a business (even if they're true). Even famously open-minded Google ditched Mark Jen (<http://roseandsnail.com>) after he blogged a few choice words about a Google sales conference that he claimed resembled a drunken frat party. The notable part of his story is that he didn't set out to undermine Google or publicize his blog. In fact, only his close friends and family even knew he had a blog. Unfortunately, a few Google-watching sites picked up on the post and sent the link around the Internet.

There are many more stories like these, where employees lose their jobs after revealing trade secrets, admitting to

inappropriate on-the-job conduct (for example, posting risqué at-work photos or bragging about time-wasting games of computer solitaire), or just complaining about the boss.

To protect yourself from the hazards of blogging, remember these rules:

- “Anonymous” never is.
- If you plan to hide your identity, adopt a pseudonym, or conceal personal details, remember the first rule.
- Funny is in the mind of the beholder. Your humorous work-related stories will be seen in a different light when read by high-powered executives who lack your finely developed sense of irony.
- Think before you write. There's a fine line between company secrets and information in the public domain.
- There's no going back. Although many blogging tools let you edit or remove old posts, the original versions can stick around in search-engine caches for eternity (see page 299).

Syndication

One of neatest features of blogs is *syndication*, which lets avid blog readers monitor their favorite blogs using a program called a *feed reader* or *news aggregator*. To use a feed reader, you enter links to all your favorite blogs, and then keep an eye out for updates. The feed reader periodically checks these blogs and alerts you to new postings, saving you from having to check every blog 94 times a day to see if there's fresh content. If you follow blogs regularly, feed readers are the most practical way to stay current with all your friends in the blogosphere.

Tip: Feed readers are a little like email programs, which, of course, regularly check for new messages from your friends. This is a lot more efficient than contacting each of them and asking if they have anything new to say. Similarly, you can use a feed reader to monitor friends' blog activity. If there's nothing new, you find out in an instant.

Although most blogs work with feed readers, some don't. To work with a reader, blogs need to provide a *feed* (Figure 13-2), a computer-friendly format of recent blog postings. Feed readers interpret the feeds and cull important information from them, like the post's title, description, date, and text. They display that information for your reading pleasure, without forcing you to make a separate trip to the blog website.



The screenshot shows a Mozilla Firefox browser window displaying the homepage of [The Pioneer Woman Cooks](http://thepioneerwoman.com/cooking/). The page features a large banner for February 2011, a navigation menu with categories like SITIONS, COOKING, PHOTOGRAPHY, HOME & GARDEN, HOMESCHOOLING, and TASTY KITCHEN, and a search bar. A prominent orange circular icon with a white 'RSS' symbol is circled, indicating the feed link. Below the banner, there's a section for 'Breakfast' recipes featuring a photo of a pancake. On the right side, there's a bio for Ree Drummond and links to social media and contact information.

Figure 13-2:
Most blogs have a feed link somewhere on their home page. Look for the word "feed," "syndication," "subscribe," or "RSS" (which is the actual format that feeds use). Usually, the link will have the orange "radar" icon shown here.

If you want to try a feed reader, you've got lots of choices:

- Online feed readers require that you sign up and create a free account, but you don't need to install anything on your computer. You just go to the feed reader website and read the feeds right in your browser window. Popular examples include Google Reader (www.google.com/reader) and NewsGator (www.news-gator.com).
- Desktop feed readers run on your desktop computer. You can check out and download many of them from popular shareware sites like www.download.com. If you use a Windows computer, you can download the excellent FeedDemon at www.feeddemon.com. Mac fans might like the highly touted NetNewsWire at <http://netnewswireapp.com>. Both programs are free.

- Web browsers are increasingly adding features like feed readers. If you use Firefox, for example, its *live bookmarks* feature tracks feeds (see Figure 13-3). Internet Explorer 7 (and later) and Safari also have built-in readers. (For more information about IE's reader, check out www.microsoft.com/windows/rss. For Safari, read the overview at www.apple.com/safari/features.html.) However, no browser offers feed-reading features as slick or convenient as dedicated programs like FeedDemon.

**Figure 13-3:**

Top: When Firefox detects a link to a feed in the current page, it displays a special orange icon in the address bar. Click this icon, and then choose one of the "Subscribe" options to add a live bookmark.

Middle: Firefox asks you how you want to store your bookmark. You can use Firefox's Live Bookmarks feature or a separate feed-reading service like Google Reader (just pick it from the list). Click the "Always use" setting if you don't want to see this feed options page again. Then click the Subscribe Now button.

Bottom: Live Bookmarks provide a submenu of current blog posts, which Firefox updates automatically. You still need to check the bookmark to see if there's a new post (which requires more effort than a feed reader like FeedDemon), but you don't need to keep visiting the original site.

Tip: Most blogs let you subscribe by email, which means you get new postings delivered to your inbox. If you follow just a few blogs and they aren't too busy, email subscriptions can avoid the hassle of setting up a feed reader.

Blog Hosting and Software

Before you set up your own blog, it helps to understand the different kinds of blog-making options out there. There are really two types of blog:

- **Self-hosted blogs.** If you're a high-tech geek with a bit of ambition, you might be interested in hosting a blog entirely on your own. To do this, you need to pick the blogging software you want, find a web host that supports it, and then configure everything. This approach gives you unlimited flexibility (and possibly better performance). If you enjoy do-it-yourself challenges like making your laptop talk to your coffee maker, a self-hosted blog is a good choice.

Examples of blogging software include WordPress (<http://wordpress.org>) and Movable Type (www.movabletype.org).

- **Hosted blogs.** With a hosted blog, you simply sign up with a blog provider and start blogging away. Adding a blog entry is as simple as filling out a form in your browser. You never need to hassle with a separate program or figure out how to upload content files, because the blog provider stores all your files for you. You don't even need to have a website. Hosted blogs are the best bet for new bloggers, because they're completely painless and remarkably flexible.

Examples of hosted blog providers include Blogger (www.blogger.com), WordPress (<http://wordpress.com>, not <http://wordpress.org>, which is where you get the free blog-hosting program), and TypePad (www.typepad.com).

In this chapter, you'll spend your time using one blogging tool, called Blogger. Blogger is tremendously popular—in fact, it has two entries in the top 20 most-visited websites (<http://mostpopularwebsites.net/1-50>). The advantages to Blogger are that it's simple to use yet remarkably powerful, with support for group blogging, personalized domain names, customized templates, and a variety of add-on features. These attributes make Blogger the best candidate for all-around blogging champ.

Note: A number of website-creation tools called *content management systems* (CMS) let you build ordinary blogs *and* other types of websites. Two popular examples are Drupal (<http://drupal.org>) and Joomla (www.joomla.org). Typically, these products suit businesspeople who need to set up complex web applications—say, a sprawling e-commerce store or a web magazine with a team of contributors—without building everything from scratch.

POWER USERS' CLINIC

Why You Might Not Want Blogger

Some die-hard blogging fanatics prefer self-hosted blogs, which offer a few unique advantages that a Blogger can't match. Here are some examples:

- **Censorship-proof.** When you use a free blogging service like Blogger, Google is in control of your content. If other users complain that your blog has offensive content or pornography, Google can force prospective readers to click through a warning page before they reach your posts. And if Google decides that you're breaking copyright laws or encouraging criminal activity, they can vaporize your blog without warning. Furthermore, some censorship-crazy countries like China block Blogger's sites, which means that no one in those countries can see your blog (unless you pay for a custom domain name, as described on page 372).
- **Linked websites.** Thanks to HTML's linking power, it's easy enough to send readers from your website to your blog and back again. They don't even need to know that they're changing web servers. However, if you host both your website and blog on the same server, you gain some unique advantages. They can

use the same domain name, they can share content (like pictures), and they can benefit each other in the search engine rankings because Google sees them as part of the same site.

- **Complete control.** Some blogs involve multiple authors and huge amounts of traffic. Their scope and popularity rival traditional newspapers. (See the Huffington Post, at www.thehuffingtonpost.com, for the most popular example.) The creators of this sort of site need complete control over every fine detail to craft search engine campaigns and advertising strategies. Clearly, in this echelon, a free blog service just won't do.

These features don't come without a cost. Self-hosted blogs are more complicated to set up and manage, and you also need to pay a monthly hosting fee. But if you're still interested, start out with WordPress (<http://wordpress.com>), the favorite of hard-core technogeeks everywhere. And if you're still not sure, don't be afraid to embrace Blogger—or any blogging service. Your choice won't limit your success, as there are mind-blowingly popular blogs on all the blogging platforms mentioned in this chapter.

Getting Started with Blogger

Blogger is one of the most commonly used blogging services. It provides the easiest way to start a blog, and it's chock full of nifty blog management tools. Once upon a time, Blogger's premium features required a small yearly contribution. But all that changed when Google bought Blogger. Now all of Blogger's features are part of the same free package.

Setting up a blog on Blogger is ridiculously easy. In the following sections, you'll learn how to create a blog, add posts, and take charge of a few neat features.

Tip: You can also check out the official catalog of Blogger help at <http://help.blogger.com> and the discussion boards at www.bloggerforum.com, where bloggers share tips, ask questions, and vent their frustrations.

Creating a Blog

Before you create your blog, it's a good idea to assess your goals and decide exactly what type of content you plan to showcase. Although you can create a blog with random thoughts or a chronicle of daily life, the most successful blogs have a clear voice and purpose. They attract a loyal audience with targeted, topic-specific posts.

Once you know how you want to position your blog, you'll be able to choose a snappy name and a suitable URL. Start with these steps:

1. Go to www.blogger.com.

This is the home page for the Blogger service.

2. If you have a Google account, enter your login information and click Sign In. If not, click "Get started."

Before you can create a blog, you need a Google account (see Figure 13-4). If you use other Google services (like Gmail, Google AdSense, or Google Analytics), you can use your existing account to create your blog.

The screenshot shows a Mozilla Firefox browser window with the title 'Blogger: Create Blogger Account - Mozilla Firefox'. The address bar shows 'google.com https://www.google.com/accounts/NewAccount?service=blogger&continue=http'. The main content is a 'Create a Google Account' form. It has a step indicator '1 Create a Google Account' and the Google logo. The form fields include:

- Email address (must already exist): mmacdona3242@qlink.com
- Retype email address: mmacdona3242@qlink.com
- Enter a password: (redacted)
- Retype password: (redacted)
- Display name: Matthew MacDonald

On the right, there are explanatory notes and validation messages: 'You'll use this address to log in to Blogger and other Google services. We'll never share it with third parties without your permission.', 'Type in your email address again to make sure there are no typos.', 'Must be at least 8 characters long.', and 'The name used to sign your blog posts.'

Figure 13-4:
The first step in creating a blog on Blogger is to sign up for (or sign into) a Google account.

Note: You need to create an account only once. However, you can create multiple blogs for the same account.

3. Type in your account information.

If you already have a Google account, type in your display name (which Blogger uses to sign your blog entries) and password. If you don't yet have an account, you need to supply your current email address, the password you want to use, and your display name.

Either way, you'll also need to type in a string of letters to prove you're not a computer program before you continue, and turn on a checkbox at the bottom of the page to officially accept Blogger's rules.

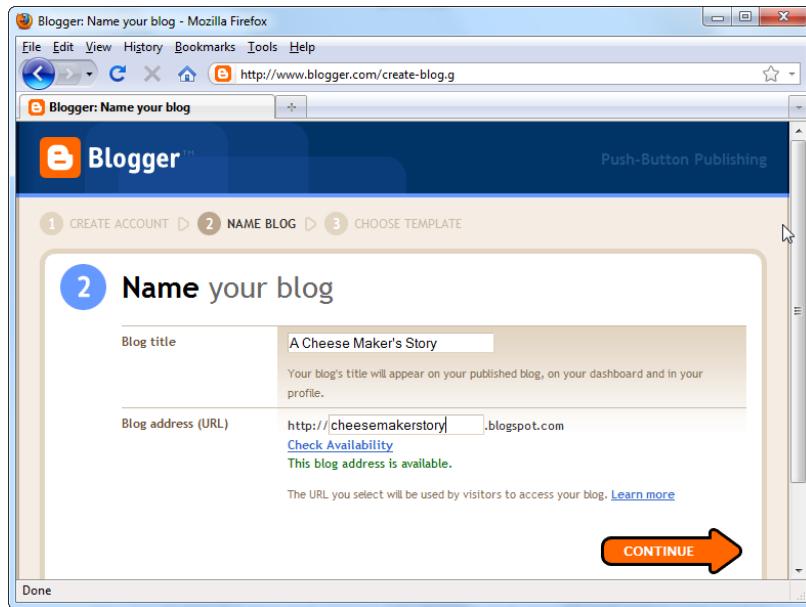
Note: At some point in the blog-creation process, Google may ask you to confirm your identity. To do this, it offers to text or phone you with a confirmation code. Passing this test is easy. First, give Google your phone number. Then, when Google calls you seconds later, type the confirmation code into the sign-up page (where Blogger is patiently waiting) and click Verify.

4. Click Continue to move to the Blogger Dashboard.

The Blogger Dashboard lists all your blogs and lets you manage them.

5. Click Create Your Blog Now to move to the next phase of the blog creation process.

Blogger displays a new page (see Figure 13-5).



The screenshot shows a Mozilla Firefox browser window with the URL <http://www.blogger.com/create-blog.g>. The page is titled 'Blogger: Name your blog'. It displays the second step of a three-step process: 'Name your blog'. The steps are numbered 1, 2, and 3: 1. CREATE ACCOUNT, 2. NAME BLOG, 3. CHOOSE TEMPLATE. Step 2 is highlighted with a blue circle containing the number 2. The form fields show 'Blog title' as 'A Cheese Maker's Story' and 'Blog address (URL)' as '<http://cheesemakerstory.blogspot.com>'. A note below the URL says 'Check Availability' and 'This blog address is available.' An orange 'CONTINUE' button is at the bottom right. To the right of the screenshot, a vertical text block provides context for Figure 13-5.

Figure 13-5:
In the second step, you create the important part—the blog itself. To make sure you're in the clear, click Check Availability to see if your blog URL is available. (If not, you need to try again.) Fortunately, it's easier to find a catchy Blogger URL (which always ends with .blogspot.com) than it is to find a good website URL.

6. Supply the title and URL you want your blog to have.

A blog title is just like a web page title. It's the descriptive bit of text that appears in a browser title bar.

The URL is the really important part, because you don't want to change this later on and risk losing your loyal readers. It's the address that eager web followers use to find your blog. Blogger is surprisingly generous with URLs. Unlike free web hosting providers, Blogger lets you use just about any URL, so long as it ends with `.blogspot.com`. Although other bloggers have already taken some of the most obvious names, it's still reasonably easy to create short-and-sweet blog names like `http://secretideas.blogspot.com` or `http://richwildman.blogspot.com`.

If you want to use a custom domain name for your blog, you need to add it *after* you create your blog with an ordinary `.blogspot` address. Page 372 explains how.

7. Click Continue, and choose a template for your blog.

Blogger gives you just a few starter templates to choose from (see Figure 13-6). But fear not—once you create your blog, you can modify your template's formatting or swap in something completely different (373).

The screenshot shows a Firefox browser window with the title 'Blogger: Choose a template - Mozilla Firefox'. The URL in the address bar is `http://www.blogger.com/choose-template-new.g?blogID=7558312199526328468`. The main content area is titled '2 Choose a starter template' with the sub-instruction 'You can change your template later, and even customize it with the Template Designer.' Below this, there are four template preview cards: 'Simple' by Josh Peterson (highlighted with an orange border), 'Picture Window' by Josh Peterson, 'Awesome Inc.' by Tina Chen, and 'Watermark' by Josh Peterson. At the bottom left is a 'Done' button.

Figure 13-6:
When you create a blog, you choose a preset template, and Blogger formats your posts with the template's color, graphics, and layout. If you change your template later on, Blogger adjusts all your posts to match the new visual style.

8. Select the template you want, and then click Continue to finalize your blog.

Blogger displays a congratulatory message on creating your brand-spanking-new blog.

9. Click Start Blogging to create your first blog post.

You can return to manage your blog any time by going to www.blogger.com. For now, continue with the next step to create your first blog entry.

10. Enter the title for your blog post, and then type the content of your post into the large text box, which acts like a miniature word processor (see Figure 13-7).

Don't worry about all the fancy frills in the editing window just yet—you'll learn about those in the next section.

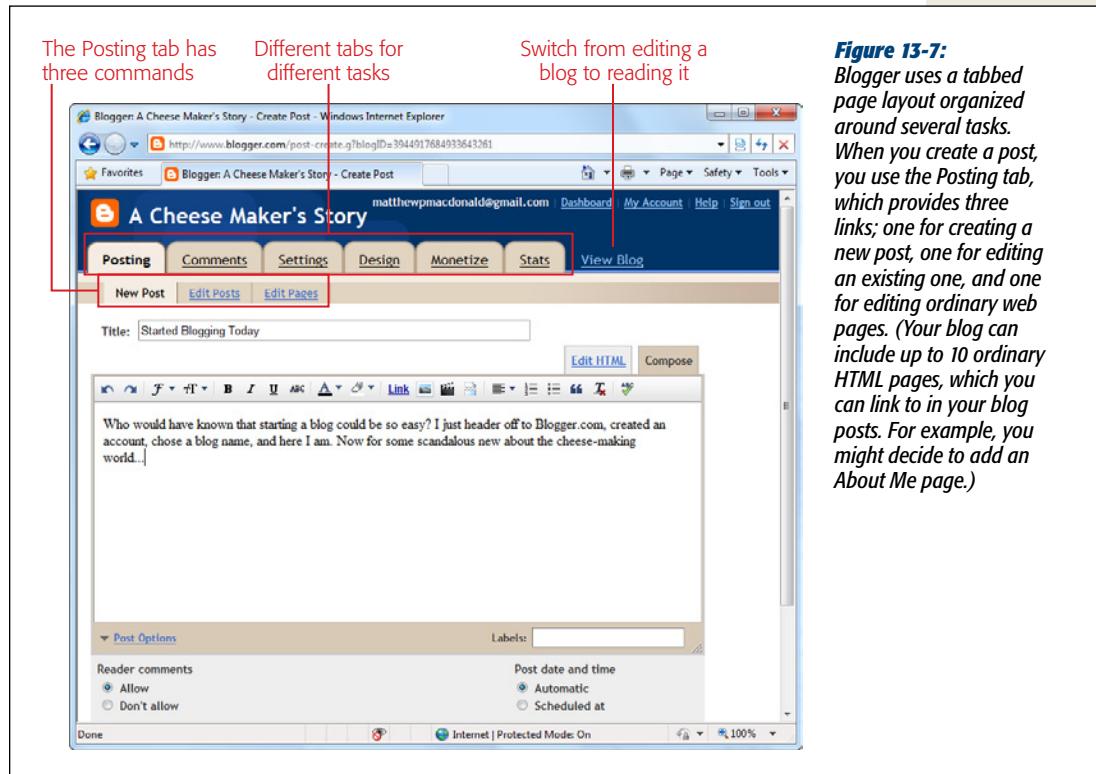


Figure 13-7:
Blogger uses a tabbed page layout organized around several tasks. When you create a post, you use the *Posting* tab, which provides three links; one for creating a new post, one for editing an existing one, and one for editing ordinary web pages. (Your blog can include up to 10 ordinary HTML pages, which you can link to in your blog posts. For example, you might decide to add an *About Me* page.)

Note: A blog entry can be as long or as short as you want. Some people blog lengthy stories, while others post one or two sentences that simply provide a link to an interesting news item (or, more commonly, to a post from another blogger).

11. Click the Post Options link at the bottom of the page to set a few more options.

You can categorize your post by filling in a keyword (or a list of keywords separated by commas) in the “Labels for this post” box. When people run a search on your blog, they can look for posts with specific keywords. For best results, always use the same keywords to identify the same things. For example, every time you talk about your pet hogs, add the label *pig*.

You can let people comment on your posts (choose Allow, the automatic setting), or you can prevent them from doing so (choose “Don’t allow”).

You can set the post date and post time. Choose a date in the future, and Blogger waits until then to add your post to the blog.

12. Click Publish Post to create your blog entry.

Blogger displays a confirmation message, informing you that it posted your new entry.

If, instead of publishing right away, you want to take some time to think over your post, click Save Now. That way, Blogger saves the text you just entered, and keeps it waiting for you the next time you return to your blog. (Page 367 explains how you can find an unposted entry and edit it.)

Tip: Blogger automatically saves a draft of your post as you type, just in case you run into Internet troubles (or you accidentally close the browser window). However, it saves your entry only every few minutes. Clicking Save Now saves your current draft immediately.

Now is a great time to check out what your post looks like. Click the View Blog link, or type your blog URL into a browser by hand. Figure 13-8 shows an example of what you’ll see.



Figure 13-8:
This blog shows two recent posts (in reverse chronological order, so the most recent update appears first). On the right, a sidebar provides sections of information about the author and other websites of interest (neither of which has been filled in yet), along with links to recent posts.

Creating Formatted Posts

So far, you've seen how to post text-only content in a blog. But Blogger's pretty flexible when it comes to customizing your blog. You can implement all sorts of fancy design maneuvers, from highlighting text to inserting graphics. Best of all, Blogger lets you run rampant with the HTML markup. You just need to know your way around the Blogger editor.

To do some customizing, start a new post by clicking the Posting tab. Type something in the Compose box in the middle of the page. Next, select some text and try out some of the buttons in the toolbar above the text box (see Figure 13-9). Behind the scenes, Blogger uses inline styles (page 137) to format your post.

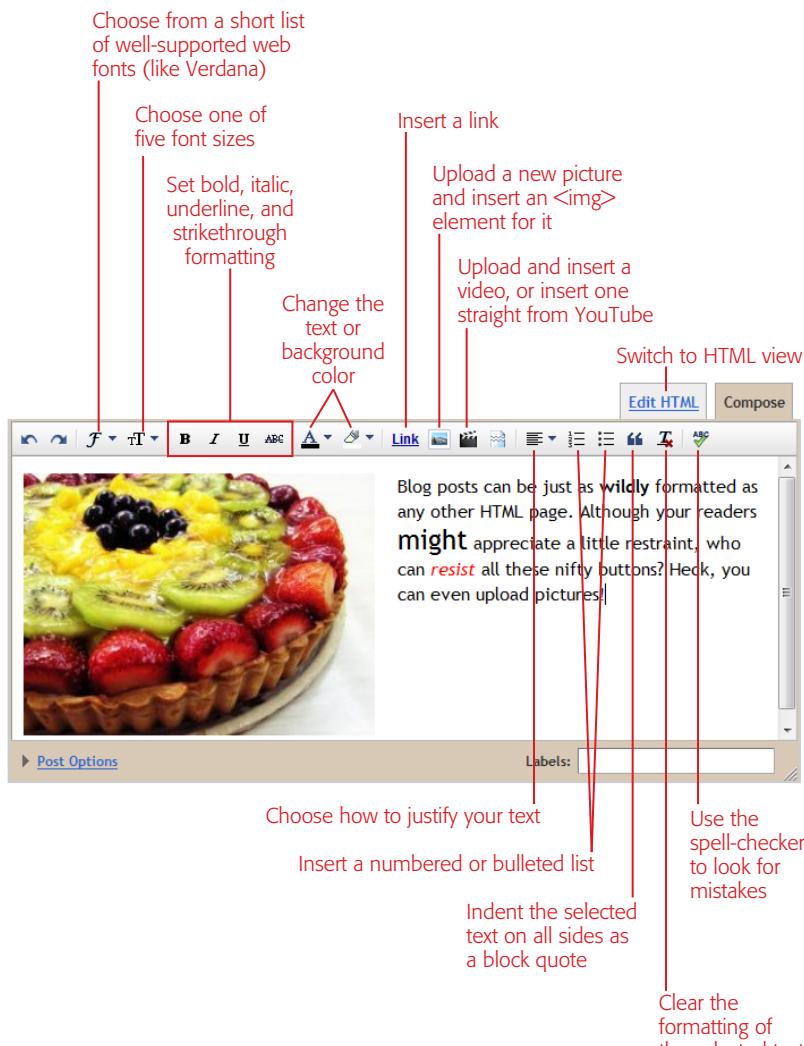


Figure 13-9:
The toolbar buttons in Compose mode limit you to a few basic choices. You can change your post's font, resize the text, add bold or *italic* formatting, create simple lists, and add pictures.

This editor, called the *visual composer*, is designed to mimic a word processor. However, if you're itching for some HTML action, click the Edit HTML link at the top-right of the edit window. Now you can add elements and other HTML goodies directly. Click Compose to go back to the WYSIWYG view.

Remember, this sort of formatting is a one-off. If you want to consistently change the way common elements appear in your blog posts (for example, post titles, the blog text, the author byline, and so on), you need to modify your blog's template (page

373). This approach is akin to using a style sheet on a large website: By editing your template, you create a set of formatting rules that customizes the look of your blog.

GEM IN THE ROUGH

Emailing a Blog Entry

To take advantage of all of Blogger's features, it makes sense to use the editor on the Blogger website. However, if you find yourself on the go with only limited time to spare, you might appreciate Blogger's ability to turn email messages into blog posts. That really comes in handy if you want to email a post from a mobile phone or if you've got a sporadic Internet connection. In the latter case, you can prepare a post in your email program, and then connect to the Internet just long enough to send in the posting.

To use Blogger's email posting features, you first need to turn them on. Click the Settings tab, then click "Email & Mobile," and scroll down to the Posting Options section. In the Email Posting Address box, enter a secret word. Blogger incorporates this word into the email address it gives you for remote posting. This prevents other people from sneaking their posts onto your blog, because they won't know the exact address. For example, if your user name is lisajones and your secret word is antelope12, you need to send a message to lisajones.antelope12@blogger.com to add a post to your blog.

Underneath, choose either "Publish emails immediately" (Blogger publishes emailed posts as soon as it gets them) or "Save emails as draft posts" (Blogger converts emailed posts into drafts, and you need to log in and review them before they go live). Or, if you decide to turn off email posting later, choose Disabled. Then click the Save Settings button at the bottom of the page to make your changes official.

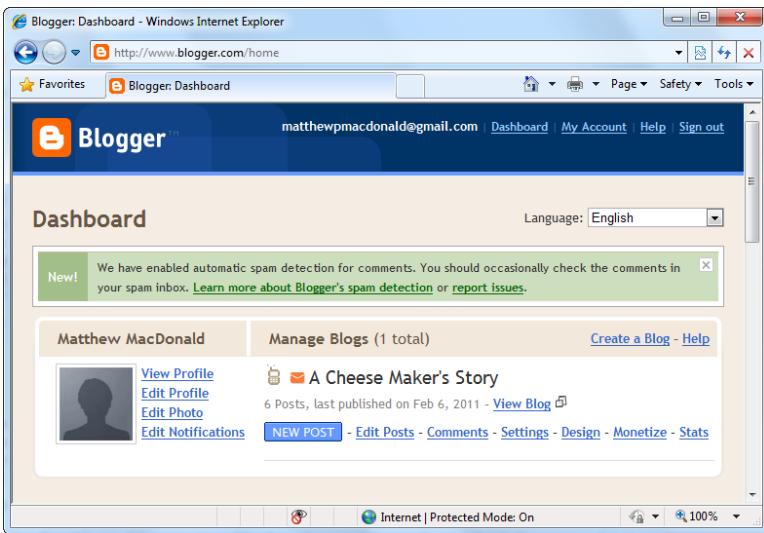
When you send an emailed post to Blogger, the subject line immediately becomes the title of your blog entry, and the message text becomes the body of the post. Best of all, you can take advantage of your email program's formatting features, because Blogger understands all the standard HTML tags. Finally, if you use an email account that adds some sort of text to the end of your message (like a confidentiality disclaimer), you can prevent that from appearing in your post. Just place the text `#end` at the end of your email message, and Blogger ignores anything that comes after.

Managing a Blog

Once you create your blog, you can perform any of the following tasks:

- Add new posts
- Edit existing posts
- Review comments left by other people
- Change your blog settings

The hub for managing your blog is a page called the Dashboard (see Figure 13-10). That's where you start every time you log in. The Dashboard lists all your blogs and provides a set of management links for each. If you scroll down the page, you'll find the "Reading List" section, which has links to blogging news, hot blogs, and any blogs you've chosen to follow.

**Figure 13-10:**

When you log in to Blogger, you start out in a section called the *Dashboard*, where you see all your blogs and how many posts they have. Click the New Post button to add a new blog entry (Figure 13-7), or click Edit Posts to review earlier entries (Figure 13-11).

Although the Dashboard is the starting point for blog management, as soon as you get to work you'll return to the tabbed Blogger page you used when you created your first post. This page is the core of the Blogger service; depending on the tab you choose, you can do anything from adding and editing posts to changing the way your blog works.

For example, if you click Edit Posts, you'll end up back on Blogger's management page, but with the Edit Posts tab selected (see Figure 13-11). This tab lets you review and edit the posts you already published. It's the perfect tool for correcting a blunder—if you're fast enough to catch it before anyone else does.

Note: If you haven't yet posted your blog entry, you see it listed with the word "Draft" next to it. This handy feature lets you take a break and pick up your blog posting the next day (or the next week), right where you left off.

There's a lot of stuff packed into this management page. Here's a quick tour of its many tabs:

- **Posting.** Use this tab to create a new post, edit an existing one, or create a few miscellaneous HTML pages (no more than 10) that can co-exist with your blog on the same site.
- **Comments.** This tab lets you review the comments that other readers add to your blog posts (and those that Blogger identifies as just random spam, and blocks. As you'll learn on page 379, you can decide which comments stay and which get removed.

Figure 13-11: The Edit Posts page lets you review what you've written. You can sift through the full list of posts, search for specific content (type something in the Search box, and then click Search), or review just those posts that have a specific label (click the label in the Labels box at the far left). Once you find the entry you want to change, click Edit. Or, if you have second thoughts about something you already posted, click Delete to remove the post altogether.

The screenshot shows the 'Edit Posts' page of a blog titled 'A Cheese Maker's Story'. The page includes a navigation bar with tabs for Posting, Comments, Settings, Design, Monetize, Stats, and View Blog. Below the navigation is a sub-navigation bar with New Post, Edit Posts (which is selected), and Edit Pages. A search bar and a 'Posts Per Page' dropdown are also present. The main content area displays a list of five posts, each with a checkbox, an 'Edit' link, a 'View' link, a timestamp, the author, and a 'Delete' link. The posts are: 'Just Spotted: Mac Wants its Cheese' (draft), 'One Pear Salad, in Search of Roquefort', 'Still Bored By Life', 'Started Blogging Today', and 'Bored By Life' (with 2 comments). Below the list are 'PUBLISH SELECTED' and 'DELETE SELECTED' buttons. Red annotations with arrows point to the 'Edit' link in the first post ('See the full post (in a new window)'), the 'Delete' link in the fifth post ('Delete this post'), and the 'Edit' link in the third post ('Edit this post').

- **Settings.** This tab groups a dizzying number of options into several subgroups. You can set everything from basic information about you and your blog to options for managing reader comments and custom domain names.
- **Design.** This tab lets you rearrange the content on your page or choose a new template. If you aren't happy with your blog's look and feel, come here to give your blog an effortless makeover.
- **Monetize.** Once your blog is established, you can capitalize on your readership by displaying them advertisements through Google's AdSense program (discussed in more detail in Chapter 14). Click your way to this tab, choose where the ads should appear, and sign up for AdSense (see page 385).
- **Stats.** With this tab you can analyze the traffic pouring (or trickling) into your blog. It uses a variation of the Google Analytics service you learned about in Chapter 11, which means you can easily pick out popular posts and the geographic areas that love you.

- **View Blog.** This link gives you a reader's view of your blog. After you make changes to your blog, you can use this link to take a look at the results.

Tweaking Common Settings

Now that your blog is up and running, take some time to fine-tune a few settings. In the following steps, you'll add a description for your blog, choose how many posts you want to display on your home page, and set the time zone to make sure your posts get the right date stamp. Along the way, you'll get a look at the many Blogger settings under your control.

1. If you're at the Dashboard, click the **Settings** link next to your blog. If you're already working in the management page, click the **Settings** tab.

You'll see several options under the **Settings** tab, each of which has its own group of suboptions. Initially, Blogger displays the **Basic** section, which includes some of the most commonly tweaked settings.

2. Add a description for your blog.

This text appears on your home page, usually just under your blog title, though the exact spot depends on your template. Try to keep the description to a sentence or two that hints at the flavor of your blog. Two good descriptions are "The sober confessions of an unlicensed meat handler," and "An on-again, off-again look at my life and adventures."

3. Scroll down, and then click **Save Settings**.

When you save your settings, your changes take effect immediately. But before you check out your blog, there's still more work to do.

4. Under the **Settings** tab, click **Archiving**.

Archiving is the process Blogger uses to group together old posts and shuffle them out of sight. Every archive gets a link on your home page. For example, if you have Blogger create monthly archives, it adds links like January 2011, February 2011, and so on. If a visitor clicks one of these links, Blogger displays the posts from that period.

5. Set the **Archive Frequency**, and choose whether or not you want each post to have its own page.

You can instruct Blogger to archive your posts monthly, weekly, daily, or not at all. Casual bloggers usually find monthly the best choice. If you blog every day, you might split posts into weekly groups, but you'll end up cluttering your index page with a lot of extra links (one for every week you blog).

6. Click **Save Settings**.

You're still not quite finished.

7. Under the **Settings** tab, click **Formatting**.

The **Formatting** options let you choose how many postings Blogger displays on your home page and how it formats dates (see Figure 13-12).

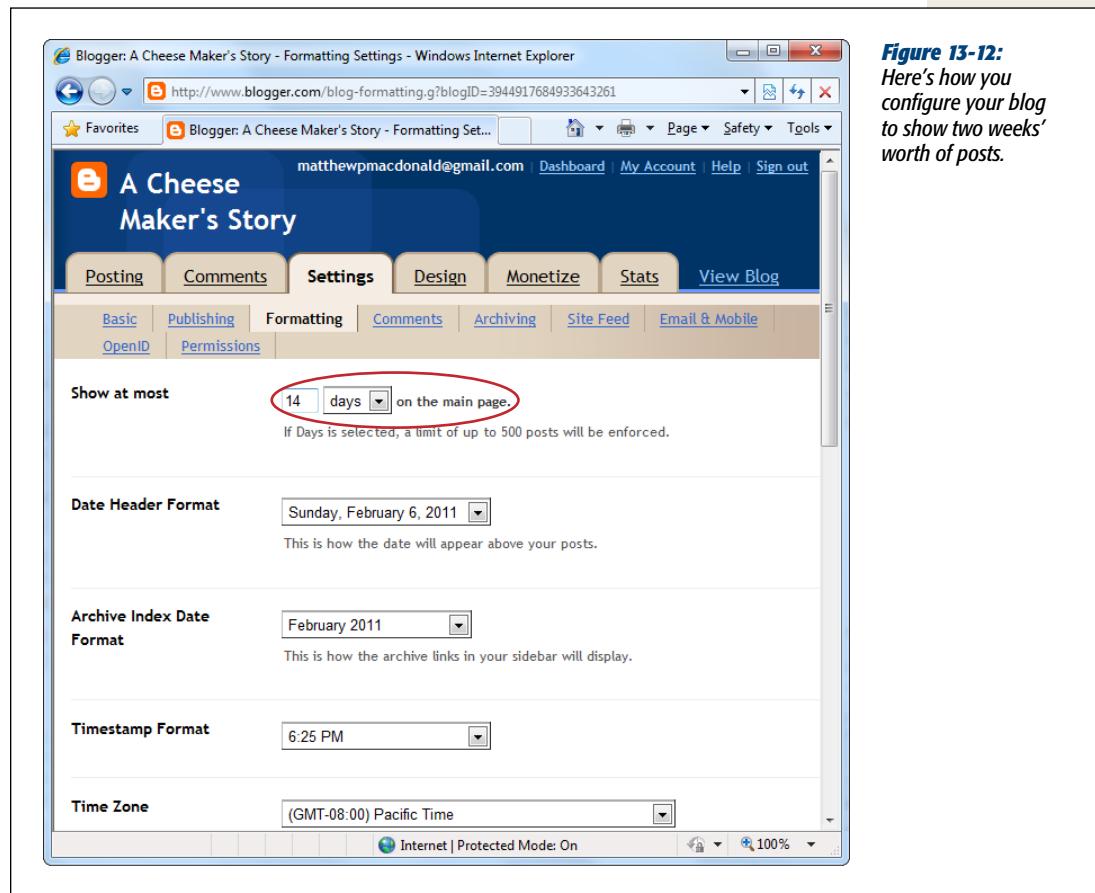


Figure 13-12:
Here's how you
configure your blog
to show two weeks'
worth of posts.

8. Choose the number of posts you want to appear on your first page.

You can ask Blogger to show a specific number of posts or number of days. For example, you could ask Blogger to show your last 14 days' worth of posts, or you could tell it to display just your three most recent posts, no matter when you published them. For best results, don't crowd your front page with too many entries. If you post daily, stick to a small number of posts or just to topics from the current week.

9. Select a date format, and specify your time zone.

Blogger dates every blog post at the beginning or end of your entry, depending on your template. By setting the time zone, you won't need to manually set the date every time you create a post.

10. Click Save Settings.

To see how these changes look, click View Blog.

GEM IN THE ROUGH

Group Blogging

Having trouble keeping your blog up to date? If you want to be part of the blogosphere but just can't manage to post more than once a month, consider sharing the effort with some friends. Look for a natural reason to band together—for example, colleagues can create blogs to discuss specific work projects, and families can use them to keep in touch (if they're not already addicted to Facebook). On a larger scale, group blogging lets like-minded people create a blog that's greater than the sum of its parts. Two wildly popular, trend-setting group blogs are Lifehacker (<http://lifehacker.com>), which posts about hacker-style productivity tricks applied to real life; and the Daily Kos (www.dailykos.com), which provides left-leaning political news and analysis.

Creating a team blog in Blogger is easy. Take your ordinary blog, choose the Settings tab, click Permissions, and then click the Add Authors button to add fellow bloggers.

You need to supply just one piece of information—the email address of the blogger you want to enlist. Blogger sends an invitation to each potential blogger. To accept the invitation, the recipient clicks a link in the email message (and creates a Google account, if the blogger-to-be doesn't have one).

All bloggers have the ability to post entries. Additionally, you can give some bloggers administrator status, which means they can add more bloggers themselves (and delete existing ones).

Configuring Your Blogger Profile

Interested in customizing the information that appears beside your posts on the home page? Blogger pulls this information from your user profile, and it's easy to customize it. Follow these steps:

1. **Head to the Dashboard on Blogger's main page.**

If you're in the tabbed view, click the Dashboard link at the top-right of the page. Otherwise, go to www.blogger.com and sign in.

2. **Click the Edit Profile link (which appears next to the head-and-shoulders silhouette).**

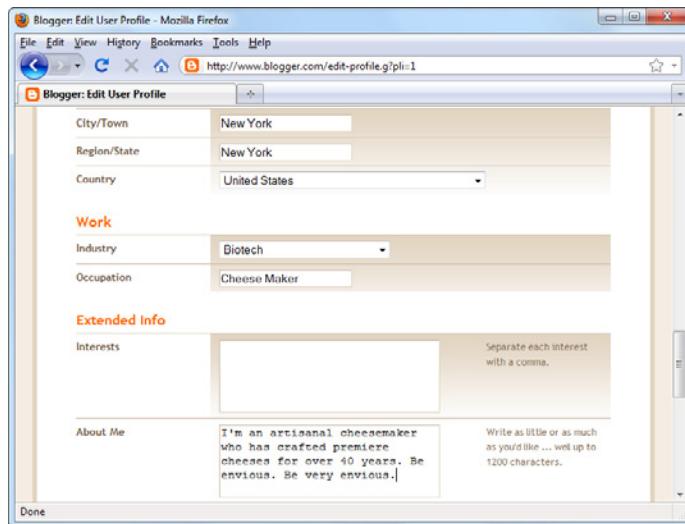
Your profile page appears.

3. **Edit your profile (see Figure 13-13).**

The profile page lets you supply a range of information about yourself. Only a few of these details appear on your home page, including your name ("Display Name"), an optional photo, your location ("City, State, and Country"), and the descriptive text in the About Me box.

4. **Once you enter all the profile information you want to supply, click Save Profile.**

Head back to Blogger's main page (click Dashboard) and click View Blog to see the results of your changes. Figure 13-14 shows an example.

**Figure 13-13:**

The *About Me* section is one of the most important parts of your profile, because it appears prominently on your blog home page. Other sections, like *Interests*, *Occupation*, and *Favorite Movies*, don't show up on your home page, but readers can find them by clicking the *View My Complete Profile* link at the bottom of the *About Me* box.

The blog home page features a large header image with the title 'A Cheese Maker's Story'. Below the header, a post from 'Sunday, February 6, 2011' is displayed with the title 'Still Bored By Life' and a snippet of text about cheese. Another post from 'Tuesday, February 1, 2011' is partially visible with the title 'Started Blogging Today'. To the right, a sidebar contains the 'About Me' section, which includes a photo of a man, his name 'Matthew MacDonald', his location 'New York, New York, United States', and his bio: 'I'm an artisanal cheesemaker who has crafted premiere cheeses for over 40 years. Be envious. Be very envious.' A red box highlights the 'About Me' section, and red arrows point to the photo, the text 'About Me', 'City, State, and Country', and 'Display Name'.

Figure 13-14:

Here's a fine-tuned blog home page that displays a custom description and *About Me* text.

Giving Your Blog a Custom Domain Name

Even though there's nothing wrong with a *.blogspot.com* URL, there's a good reason to get a custom domain name for your blog. No matter how much you love Blogger right now, someday you might move on to a different service. It's always easier to make this transition if you don't need to tell all your readers to update their bookmarks and head to a completely new web address.

There are three ways to get a custom domain for your blog:

- **Buy a domain from Blogger.** This is the simplest approach.
- **Buy a domain from another domain seller.** Or, you can use a domain name you've already bought, but aren't using for anything else. Either way, the goal is to configure your domain name to point to Blogger's web servers (see step 2 below). This approach gives you more flexibility, but also requires a configuration process that you need to work out with your domain name seller.
- **Share a domain with a website you already own.** For example, if you have a website at *www.deviousweevils.org* you might want to host your blog at the subdomain *blog.deviousweevils.org*. Not all web hosts support this feature. Unfortunately, it's not possible to connect a subfolder of your website to Blogger (so give up your dreams of having your blog appear at *www.deviousweevils.org/blog*). For that, you need a self-hosted blog (page 365).

Note: Up until the middle of 2010, Blogger provided an FTP hosting feature that allowed developers to store Blogger files on their own web servers. Although this made combining a standalone website with a blog easier, this feature never caught on with the average blogger, and Google discontinued it.

No matter which option you choose, your blog's *.blogspot.com* address remains in service, and all your files stay on Blogger's web servers. You're just creating a new way for your fans to find your blog.

Here's how to make it happen:

1. If you're using a domain from another web host, make sure you bought the domain you need. If you're planning to buy your domain name from Blogger, skip straight to step 3.

Chapter 3 has much more about the art of buying a domain name. Just remember not to pay for web hosting space, because you won't need it. If you already have a web host, you can simply buy an additional domain.

2. If you use a domain from another web host, set it up to point to your blog.

To do this, follow Blogger's instructions at <http://tinyurl.com/n2zsbl>. You'll need the help of your web host to make the changes, but the instructions cover all the technical jargon you need to know.

3. Now it's time to change Blogger's configuration settings. If you're at the Dashboard, click the Settings link next to your blog. If you're already working in

the management page, click the **Settings** tab.

This takes you to the familiar Settings page.

4. Under the Settings tab, click Publishing.

The publishing settings let you choose a new *.blogspot.com* address or configure a custom domain.

5. Click the Custom Domain link.

A text box appears where you can search for URLs and, if you find the one you want, buy it from Blogger (typically, it costs \$10 a year). If this is what you want, enter your domain name, check that it's available (click the Check Availability button), and follow the instructions to pay for it through Google Checkout.

If you'd rather use a domain of your own, skip to the next step.

6. If you already own a domain, click "Switch to advanced settings."

Blogger opens a text box where you can enter the domain name you already own.

Fill in your domain name, type in the verification word (to prove you're not some sort of automated domain-registering program), and click Save Settings to make it official.

You can now access your blog through the new domain, although the original *.blogspot.com* domain will continue to work as well.

Customizing Your Template

Templates are keenly important in Blogger. They not only reflect your blog's visual style (irreverent, serious, technical, breezy, and so on), they also determine its ingredients and how those ingredients appear on the page. Fortunately, Blogger lets you change many of your template's components. For example, you can move the About Me box to a new position, modify its appearance, or remove it entirely.

You can also add new sections, like a set of links that point to your favorite fellow bloggers, or a sidebar of targeted Google ads (page 385). You can even get more radical and replace your template entirely, even if you've been posting for years. Blogger retrofits all your old posts to the new template, so your thoughts, both old and new, remain available for eager readers.

To take control of these details, click the Design tab. You'll see three choices, each of which corresponds to a way to customize your template:

- **Template Designer.** Use this to pick a new template, or change the current template's formatting. The formatting choices are so extensive that you may never need to resort to editing the raw HTML.
- **Page Elements.** Use this to rearrange the layout of your blog, remove parts you don't want, or add new types of content. For example, you can quickly throw in a list of links, a poll for your readers, a slideshow, or a Facebook sharing link.

- **Edit HTML.** Use this to customize the raw HTML in your template. If you aren't afraid to muck around in the markup, you can do anything.

You'll see how to use all these features in the following sections.

Reformatting a template

When you first create a blog, you choose a template. Over time (or even right away), you may decide that the template no longer suits your content. Fortunately, you can tweak your blog to perfection by changing its fonts, colors, and layout settings in the template designer. Here's how:

1. Under the Design tab, click Template Designer.

This page loads a two-part page, with a pile of formatting tools at the top and a preview of what your blog will look like at the bottom (see Figure 13-15).



The screenshot shows the Blogger Template Designer interface. At the top, there's a toolbar with buttons for Back to Blogger, View Blog, Help, and APPLY TO BLOG. Below the toolbar is a sidebar with navigation links: Templates, Background, Layout, Advanced, and a Post-specific section. The main area displays a blog post titled "A CHEESE MAKER'S STORY" with a preview of the content: "Still Bored By Life" and "It's only Monday, and my week's supply of fine cheeses is already depleted. What I wouldn't give for a fine football-sized slab of emmentaler! Without it, I'm nothing." Below the preview, there's a link to "Started Blogging Today". On the right side of the interface, there are three color selection panels: Title Font, Background Color, and Border Color. The Title Font panel shows a dropdown menu with "Bentham" selected, and buttons for bold, italic, and font size (30px). The Background Color panel shows a color swatch set to "transparent" and a "Colors from this template" section with several color swatches. The Border Color panel shows a color swatch set to "#ccba9a" and a "Colors from this template" section with several color swatches. Red annotations with arrows point to these panels with the following text:

- "Choose an editing task" points to the sidebar navigation.
- "If you've chosen Advanced, the next step is to pick the part of the blog you want to format" points to the Advanced link in the sidebar.
- "Give it a new font here" points to the Title Font panel.
- "And change its color here" points to the Background Color panel.

Figure 13-15:
Ready for a change? Here, the blog has a new background, a wider single-column layout, fancy fonts (note the heading in particular), and a matching color scheme.

Blogger splits the formatting tools into several categories:

- **Templates** lets you pick a whole new template, though it limits you to the same few templates you saw when you created your blog, so there's not much point in using this option.
 - **Background** lets you choose a set of themed background colors and a background picture. For the background picture, you can upload your own creation, or you can pick from a wide selection of premade backgrounds organized into categories like "Abstract," "Food & Drink," and "Travel."
 - **Layout** lets you switch between a few basic layouts. You can have a simple, single column, you can add a sidebar on either side (or both sides) of it, or place all your gadgets at the bottom, footer area of your blog. To actually move individual bits of content around the page, use the Page Elements feature described in the next section.
 - **Adjust Widths** lets you set the maximum width (in pixels) of your blog's main column and sidebars. Blogs use adjustable layouts that shrink to fit a browser window, along with a maximum-width limit that keeps page components from growing ridiculously big. Page 252 describes this type of layout.
 - **Advanced** has the most interesting formatting controls. Start by using Blogger's scrolling list to pick the detail you want to format (Page Text, Links, Blog Title, Post, and so on), and then use the controls on the right to change its font and colors. The font options are particularly impressive. Not only do they support common web fonts like Arial, Verdana, and Georgia, they also include a library of free Google fonts, implemented using CSS's hot new embedded fonts feature (page 165).
2. Click "Apply to Blog" to make your changes permanent (at least until you need your next formatting fix).
- Or, if you don't like the changes you made, click "Back to Blogger" to abandon them.

Changing (and rearranging) the gadgets in your template

The focus of your blog is the series of posts that runs down the middle of the page. However, every blog includes extra blocks of content, positioned in a sidebar next to your posts or in a footer at the bottom of the page. Examples include an About Me box, a Blog Archive box (which lists your most recent posts, organized in a tree by month), and the Google Friends list (which lists the people following your blog). Blogger calls these ingredients *gadgets*, and it gives you complete control to move them, add new ones, or remove existing ones. Here's how:

1. Under the Design tab, click Page Elements.

Blogger displays an outline of your blog's current structure, showing you the location of your current gadgets (Figure 13-16).

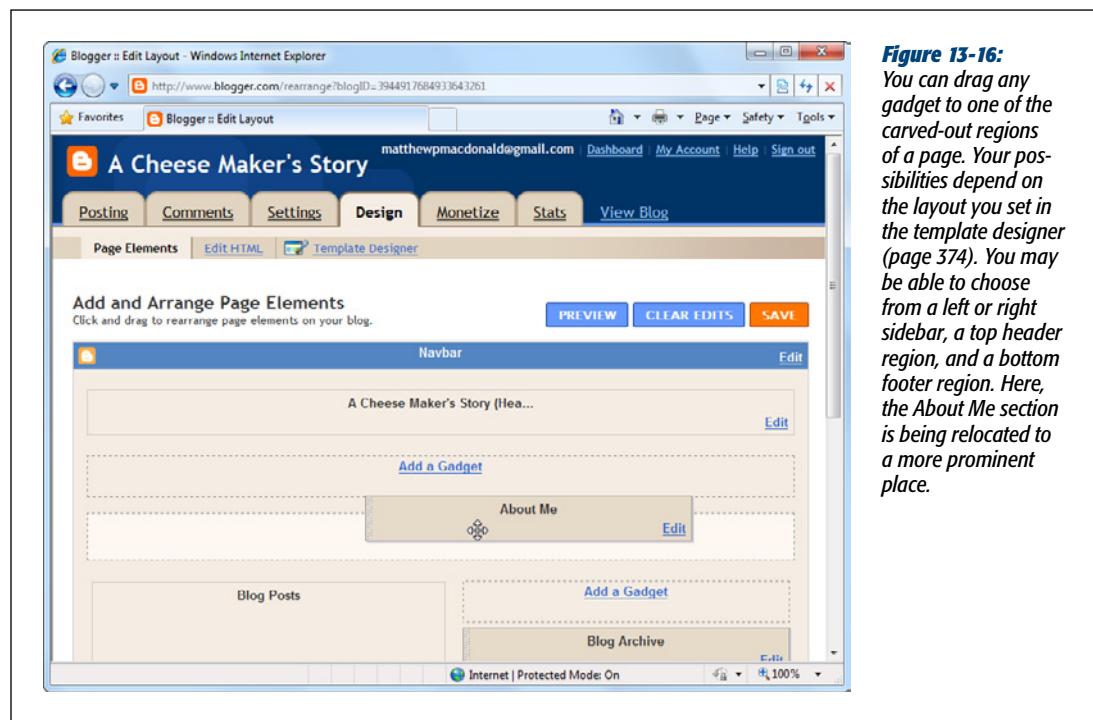


Figure 13-16:
You can drag any gadget to one of the carved-out regions of a page. Your possibilities depend on the layout you set in the template designer (page 374). You may be able to choose from a left or right sidebar, a top header region, or a bottom footer region. Here, the About Me section is being relocated to a more prominent place.

2. Move your gadgets around.

Initially, your page displays whatever content blocks your template defines. However, you have plenty of options to mix things up. The easiest change you can make is to move a gadget. Simply drag it to a new place (Figure 13-16).

3. If you want new gadgets, add them.

To add a new gadget, click one of the “Add a Gadget” links in the Page Elements preview. You see a pop-up list with Blogger’s wide choice of handy add-ons. Click one, and you see a preview of what it looks like. If you like what you see, click the plus (+) button to pop it into your page, and then drag it to any spot you want.

Some gadgets are plain—blocks of ordinary text, lists, pictures, and links, for example (see Figure 13-17). But others are much more interesting. Here are some good examples:

- **Blog List** lets you add a catalog of blogs you follow or admire, complete with links. Link List does the same thing, but with ordinary websites.
- **Search Box** adds a Google search box visitors can use to search the posts in your blog.
- **Popular Posts** displays a list of the most visited posts on your blog, so new readers can find them quickly and check them out. This feature is wildly

popular on news sites. Recent Comments is similar, but it displays the most recently left comments for any post. It lets readers quickly see what others are saying.

- **AdSense** displays the money-making Google ads described on page 385. Before you can use this gadget, you need to switch on AdSense by answering a couple of questions on the Monetize tab.
- **Twitter Updates** displays your recent tweets, if you use Twitter (page 327).
- **Flickr Photostream and Picasa Photostream** show pictures you recently uploaded to Flickr or Picasa.
- **Poll** lets you survey your readers and tally their votes.



Figure 13-17:
This blog has three new gadgets on the right: an ad box, a list of links, and a list of popular posts.

4. Configure or remove your existing gadgets.

To change a gadget's settings, click Edit in the gadget box. Your options depend on the gadget. For example, the Blog Archive gadget displays a calendar of previous blog entries. You can change the order of the entries, the way Blogger groups the posts, or the way it formats the dates. If you edit the Blog List gadget, you can supply the list of blogs you want to link to.

To remove a gadget, click the Edit link, and then click Remove. One detail you can't remove is the *NavBar*, the thin strip that appears at the top of your blog.

Visitors who view your blog can use the NavBar to travel from one blog to another, sign up for their own blog, or (most usefully) search your blog for keywords. The good news is that, although you can't remove the NavBar, you can assign it a color so that it matches your template and blends in with the scenery.

Tip: If you really, really want to remove the NavBar, you can edit the HTML of your template, and modify its style so that it's hidden. You can find this workaround described at <http://tinyurl.com/4tbxvl8>.

5. When you finish adding and arranging gadgets, click Save.

Alternatively, you can click Preview to see what your blog will look like when you commit to the changes, or click Clear Edits to abandon your changes and go back to the way things were.

Editing the HTML in a template

The tools you've used so far give you a lot of control over your blog's appearance. But there's one more frontier for die-hard bloggers who want unrestricted control over their pages—your template's HTML markup. Using the skills you learned throughout this book, you can change virtually anything there.

This is obviously more work than using Blogger's template designer. So why take this extra step? There are a few good reasons:

- **You want to use an entirely new template.** Blogger includes only a small set of templates for you to choose from. You can find more online (search for "blogger template").
- **You want to use advanced CSS formatting.** Blogger doesn't give you much design control beyond fonts and colors. If you want to throw something else into the mix—tweaking margins and padding, for example, or adding borders or setting background pictures—you need to dig into the template and modify its style sheet.
- **You want no-holds-barred customization.** Adding gadgets is a powerful system, but it doesn't give you the complete, fine-grained control that editing the HTML does. Of course, sometimes too much control makes life unnecessarily complicated.

Blogger's templates are really just HTML documents that define your blog pages. At first glance, this seems a little unusual—after all, a modest blog has dozens of pages, and you have only a relatively simple template! The trick is that the template defines special replaceable regions. When a visitor requests a page in your blog, Blogger starts with your template and fills in the appropriate content wherever it finds special codes.

For example, if Blogger finds this odd-looking code:

```
<title><data:blog.pageTitle /></title>
```

It replaces the highlighted element with your blog's title. The final HTML file it creates for your home page actually contains this text:

```
<title>A Cheese Maker's Story</title>
```

To change the HTML in your template, follow these steps:

1. In the Design tab, click Edit HTML.

You see a text box with the full HTML for your template, codes and all. You can change some details right away, like the formatting rules in the inline style sheet. But if you want to add new content or rearrange the page, you need to understand Blogger's template codes. You can get that information at <http://tinyurl.com/295vg5>.

Tip: Once you perfect your template, back it up before you make any more changes. Otherwise, you could muck it up and have no way to get back to the right version. To make a backup, click the Download Full Template link, and then pick a save location on your computer. To restore a template, click Browse to find the backup on your computer, and then click Upload to transfer it to Blogger.

2. When you finish making your changes, click Save Template.

Blogger updates your blog to use the new template immediately.

Managing Comments

Ordinarily, Blogger lets visitors leave comments on your blog. That means your readers can post their own thoughts and follow-ups, and Blogger displays them along with your posts.

To leave a comment, visitors must have a Google account or an OpenID account (a standard that many blogging services and websites use). Blogger imposes this restriction to reduce *comment spam*—distracting comments, usually posted by automated programs, that advertise the spammer's products. Fortunately, Blogger has your back. It intercepts comments that look like spam, and moves them to a special holding zone. It's then up to you to review these comments and decide whether to let them through. To do that, click the Comments tab, and then click Spam. Blogger shows a list of quarantined comments from all your blog posts. If you find a valid comment, select it and click Not Spam. And if you're drowning in junk comments, click Select All, and then click Delete to clear away the clutter.

The comment spam feature isn't as good at catching objectionable comments left by real-life people. If your blog discusses controversial topics, you might want to impose stricter regulation to prevent unhinged commenters from attacking each other. You can wrangle comments two ways: by deleting them or by moderating them.

Deleting comments

This first approach is to delete objectionable comments after the fact. To do this, simply click the Comments tab. Blogger shows a list of all the comments left on all

your blog posts, in reverse chronological order (so new comments top the list, no matter how old the post is). If you see a comment you don't like, you have three ways to dispatch it.

- **Delete** removes it immediately.
- **Spam** deletes it immediately and alerts Blogger that a spammer is at work (which may get the commenter's ID blacklisted)
- **Remove** content deletes the text of the comment, but leaves the record of the comment on the blog post, with the user's ID and the message "This post has been removed by the author." Sometimes, this approach can shame unruly commenters into better behavior, and it prevents people from criticizing you for sneaky censorship.

The only problem with deleting comments is that you have to log on regularly, and your edits take place after the fact. If you don't have the time to keep checking your blog or you want to make sure no one gets a chance to post inflammatory remarks (even briefly), you need a different approach.

Moderating comments

Your other option is to *moderate* the comments, allowing them on your blog only after you give them your personal thumbs up. Moderating comments imposes extra work, but in some situations it's the only way to lock out undesirable comments—particularly if you have a popular blog on a hot topic. Here's how you moderate comments:

1. If you're at the Dashboard, click the Settings link next to your blog. If you're already working in the management page, click the Settings tab.
2. Under the Settings tab, click Comments.

Blogger gives you a surprisingly thorough set of options to control how comments work. For example, you can control whether anonymous readers can post comments (ordinarily, they can't), or you can switch off comments entirely. But the most interesting option is the one that controls moderation.

Note: Regardless of whether you allow anonymous readers and whether you use comment moderation, you should always keep the "Show word verification" option switched on. This forces readers to type in a word from a picture before they can post their comments. Annoying as it sometimes is, this technique cripples automated comment-posting software that leaves the worst comment spam.

3. Change the "Comment moderation" setting from Never to Always or "Only on posts older than 14 days."

The "Only on posts older than 14 days" option is an interesting compromise. It lets readers post comments when your blog entry is new, which is when most comments come in, but it switches to moderated comments after a certain number of days. This model works well because spammers are far more likely to

comment on old postings than your readers are. Although Blogger suggests a timeframe of 14 days, you can reduce that number to fight comment spam more aggressively or increase the timeframe to give your readers more time to speak their mind without hassle.

Optionally, supply your email address so Blogger can notify you when someone posts a comment. However, this step isn't necessary, because you can moderate comments from Blogger's Posting tab.

4. Click Save Settings to store your new comment settings.

Now you're ready to try the system out. First, log out of Blogger. To test the moderating system, sign back in with a different Google account or as an anonymous visitor (if your blog lets anonymous readers leave comments). You could leave a comment while you're logged in as the blog owner, but in that case Blogger assumes you don't need to second-guess yourself, and it skips the moderation step.

5. Post a new comment.

To leave a comment, click the Comment link at the end of a blog entry. Add a comment and your ID information, if required, and then click Publish Your Comment to make it official. Blogger is smart enough to know that the blog uses moderation, so it displays a message explaining that the comment won't appear until the blog's owner (that's you) approves it.

6. Review your comments.

To look at new comments, log in as the blog owner again, click the Comments tab, and then click the Awaiting Moderation link underneath. You'll see the comments that await your approval (Figure 13-18).

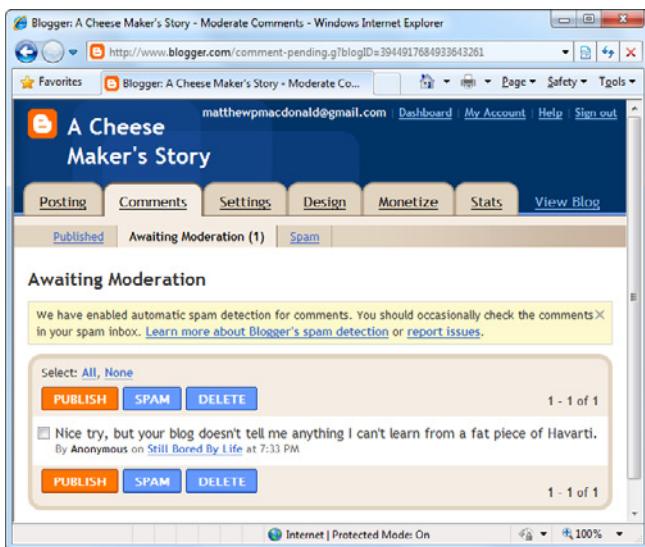


Figure 13-18:
This blog has just a single unreviewed comment. Click Publish to post it alongside the relevant post, or Delete to toss it out. If you want to approve or reject multiple comments at once, just tick the checkbox next to each one, and then click Publish or Delete.

GEM IN THE ROUGH

Promoting Your Blog

You need to promote your blog just as you do any other website. Although you can use all the techniques you learned in Chapter 11, there are some others unique to the blogosphere. Here are some important tips to get you started:

- **Add a blogroll to your site.** A blogroll is really just a set of links that lead to blogs you like. But blogrolls also make a statement. They say, “These are the people I like” or “This is the crowd I want to be associated with.” In other words, blogrolls represent social networking at its best. To use a blogroll, add a Blog List gadget.
- **Participate with others.** Bloggers are an open-minded bunch. If you leave an insightful comment in response to someone else’s blog entry, odds are good that at least some readers will head over to your blog to see what else you have to say.
- **Make it easy for people to share your post.** You need to capitalize on the enthusiasm of your visitors. If you blog about a truly fascinating piece of gossip or news, readers might just decide to tell all their friends about it—if you make it easy enough. To encourage this impulse, add a link that lets readers email your posts. Choose the Settings tab, click the Basic link, and then set “Show Email Post links” to Yes. You can also add the Share It gadget, which adds links that visitors can use to quickly recommend your post on Facebook or Twitter.

- **Promote your feed.** Feeds, discussed on page 353, work with feed readers. True blog aficionados love them because they can track dozens or even hundreds of blogs at a time. Blogger’s software lets you create a feed, and it’s worth promoting your feed to your regular readers. To see your feed, click the “Subscribe to Posts (Atom)” link on your home page. To add the familiar orange radar icon, which makes this option more obvious, add the Feed gadget to your blog.
- **Use BlogThis.** A huge number of blog postings simply call attention to interesting news stories, scandalous gossip, or funny pictures that appear online. If you’re an infrequent blogger, referencing these stories is a great way to beef up your blog. Using a nifty tool called *BlogThis*, you can create a new blog entry that links to an existing web page with a single click. There are two ways to use BlogThis—you can add the Google Toolbar to Internet Explorer or Firefox, which has a button for just this purpose, or you can add a link to your Bookmark or Favorites menu that does the same thing. For the full details, check out <http://tinyurl.com/3n4hv>.

Making Money with Your Site

If it's not for sale on the Web, it's probably not for sale at all. It's no secret that the Internet is a global bazaar with more merchandise than a decade's worth of garage sales. Web surfers generate huge amounts of traffic hunting for travel discounts, discussing hot deals, and scouring eBay for bargains. So how can you get your share of web capital?

One obvious option is to sell a real, tangible product. The Internet abounds with specialty shops hawking art, jewelry, and handmade goods. But even if you have a product ready to sell, you still need a few specialized tools to transform your corner of the Web into a bustling e-commerce storefront. For example, you'll probably want a virtual *shopping cart*, which lets visitors collect items they want to buy as they browse. And when they finally head for the virtual checkout counter, you need a secure way to accept their cash—usually by way of a credit card transaction. In this chapter, you'll learn how to implement both of these features on your site using PayPal's merchant tools.

Even if you aren't looking for a place to unload your hand-crafted fishbone pencils, your website can still help fatten your wallet. In fact, just about any website can become profitable, either by selling ad space or by recommending other companies' products. In this chapter, you'll consider how to use two of the Web's most popular *affiliate programs*—Google AdSense and Amazon Associates—to collect some spare cash.

Note: Not a U.S. citizen? Don't worry—all the money-making ideas in this chapter rely on companies that provide services worldwide. Google, Amazon, and PayPal let you rake in the cash no matter where you live.

Money-Making the Web Way

The Web offers many paths to fiduciary gain. Here are some of the most popular:

- **Donations.** It sounds crazy, but some websites badger visitors for spare change. Donations might work if your site provides some truly valuable and unique content (see Figure 14-1). Otherwise, save yourself the bother. Don't be seduced by logic like "If 1,000 visitors come to my site and every one pays just 10 cents..." They won't.

Tip: If you think that your website offers some unique, practical information, and you think your visitors might be tempted to cough up a few cents in appreciation, you can use PayPal to add a Donate button (page 414). Consider adding a message like "Buy me a beer/cup of coffee" above the Donate button to make the virtual transaction feel like a real-world tip (and to emphasize that you're hoping to collect spare change, not the next payment on your car loan).

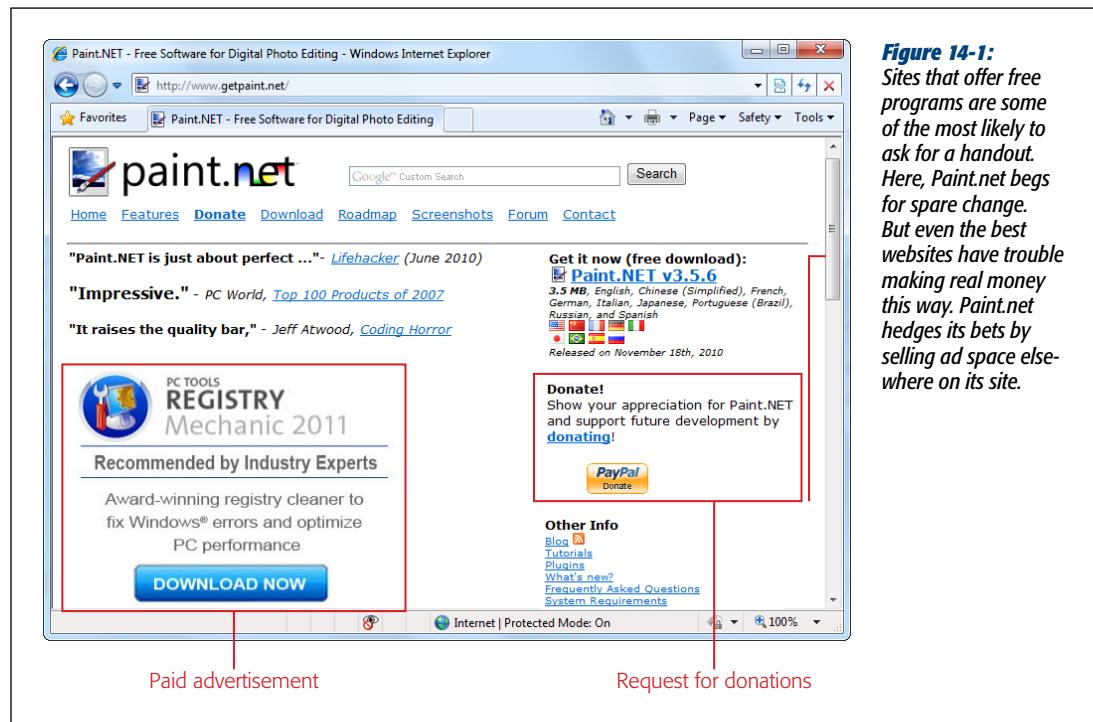


Figure 14-1:
Sites that offer free
programs are some
of the most likely to
ask for a handout.
Here, Paint.net begs
for spare change.
But even the best
websites have trouble
making real money
this way. Paint.net
hedges its bets by
selling ad space else-
where on its site.

- **Advertisements.** The most popular way to make money on the Web is by selling small pieces of web-page real estate. Unfortunately, it's also a great way to exasperate your visitors, especially if the ads are distracting, unrelated to your site, or simply take up too much space. Not long ago, ads were the worst thing you could do to web pages. Fortunately, in the 21st century, monitors are bigger,

and companies like Google provide targeted, unobtrusive ads that fit right in with the rest of your site.

- **Affiliate programs.** Rather than plaster ads across your site, why not put in a good word for a company you really believe in? Many affiliate programs give you a commission for referring customers to their sites. For example, if you review gourmet cookbooks, why not include links to those books on Amazon's website? If an interested reader buys a book, Amazon's associate program forks over a few dollars.
- **Sell stuff.** If you have your own products to sell, the Web is the perfect medium, since the cost to set up shop online is much smaller than it is in the real world. You can build a slick store, complete with product pictures and a shopping cart, with surprisingly little work. (And if you don't have your own products to sell, you can whip up some simple customized goods at CafePress, as described in the box on page 410.)
- **Pay-for-content.** If you have really great content, you can ask for cash *before* letting your visitors into your site. Warning: This is even harder to pull off than asking for donations, because visitors need to take a huge leap of faith. It's a technique used by established media companies like the *Wall Street Journal* and by hucksters promising secret ways to conquer the real estate market or to get free camcorders.

Note: Pay-for-content is the only money-making scheme you won't learn to pull off in this chapter. That's because in order for it to work, you need a way to *authenticate* visitors—in other words, you need to be able to identify visitors to tell whether they've paid you or not. This requires some heavy-duty programming (or a pay service from another company).

Google AdSense

Even if you don't have a product to sell, you still have one valuable asset: the attention of your visitors. The good news is there are a huge number of companies ready to pay for those eyes.

Some of these companies pay you a minuscule fee every time someone visits a page that carries their ad, while others pay you only when a reader actually clicks an ad, or when a visitor both clicks an ad *and* buys something. Fortunately, you don't need to waste hours checking out all these options, because Google has an advertising program called Google AdSense that handily beats just about every other system out there.

The AdSense program requires you to display small, text, image, or video advertisements on your pages. You sign up, set aside some space on one or more pages, and paste in some Google-supplied HTML (see Figure 14-2). Google takes care of the rest, filling in that space with one or more ads every time someone requests your page.

The screenshot shows a Windows Internet Explorer window displaying the website [Beyond Salmon: Whole steamed fish with ginger and scallions](http://www.beyondsalmon.com/2006/03/whole-steamed-fish-with-ginger-and.html). The page features a header with the site's name and a photo of salmon. Below the header is a sidebar with sections like 'Helen's School of Fish' and 'Recipes On The Go'. The main content area shows a recipe for 'Whole steamed fish with ginger and scallions' with a photo of the dish. To the right of the main content, there are three vertical Google AdSense ads. A red arrow points from the caption below to one of these ads. The ads have a consistent visual style with the rest of the page.

Figure 14-2:
This website nestles a box of three Google AdSense ads alongside a scrumptious recipe. The ads blend into the scenery perfectly because they have a similar visual style (they have the same background color and font), and because the content matches the article. Google calls this grouping of ads an "ad unit." You choose the ad layout and the number of ads you want per page, so it's up to you whether you want to slip a few ads in quietly or have them dominate your page.

Just displaying Google AdSense ads doesn't get you anything, but whenever a visitor *clicks* one of the ads (and travels to the advertiser's website), you earn a few cents. When your total reaches \$100, Google mails you a check or sends the cash straight to your bank account.

There's no way to know for sure how much money an individual AdSense click is worth. That's because Google advertisers compete by bidding on keywords (see page 319), and keyword prices can fluctuate over time. Google *does* let you know how much your clicks were worth (in total) when it pays you. A typical click nets you about 10 cents, but per-click prices often range from a few pennies to several dollars.

Before you become an AdSense devotee, you should know what makes AdSense different from other ad programs. Here are some of its top advantages:

- **AdSense ads are relevant.** Google automatically scans your site and picks ads based on your site's content. So if you have a site devoted to SpongeBob SquarePants, Google provides ads hawking SpongeBob DVDs, inflatable dolls, and birthday gear. Using content-based ads is far, far better than aggravating your visitors with offers for completely unrelated products, like high-tech spy cameras. Even better from a profit perspective, these "targeted" ads dramatically increase the chance that a visitor will click an ad, netting you a click-through fee. And if you're worried about a competitor's site turning up in an advertisement, you can tell Google to filter it out (see page 390).

- **AdSense ads blend in with the scenery.** Google gives you a range of layout and color options for its ads, so you can match the design and slick color scheme of your site.
- **Google provides fair payment.** As you learned in Chapter 12 (page 319), Google charges advertisers different amounts of money for different keywords. Some advertising providers pay their members the same amount for any click-through and swallow the extra money. Not Google. It pays you according to the current value of the keyword, which guarantees that you always get a competitive rate.
- **There are no start-up charges.** The AdSense program is free.

Tip: Don't try to cheat AdSense. Devious web developers have tried to game the system by clicking their own ads over and over again, or even firing up automated programs to do it for them. The problem is that Google uses various techniques to spot suspicious usage patterns. If it sees a ridiculous number of clicks over a short period of time, all originating from the same computer, it's likely to spot the deception and ban your site outright.

Signing Up for AdSense

You can learn much more about the specifics of Google's ad program by visiting www.google.com/adsense. There's also a great, not-too-detailed walkthrough at www.google.com/services/adsense_tour.

When you're ready to get started with AdSense, follow these steps:

1. On the AdSense home page (www.google.com/adsense), click the Sign Up Now button.

Google starts gathering account information (Figure 14-3). First, type in your site URL and identify its language. Next, indicate whether you're applying as an individual or as a registered business. This determines the kind of tax information Google needs to collect. Registered businesses based in the US need an EIN (Employer Identification Number). U.S. citizens applying as individuals need to give Google an SSN (Social Security Number). Citizens of other countries may need to apply for a U.S. TIN (Taxpayer Identification Number)—see www.google.com/adsense/taxinfo for the lowdown.

Note: Google won't pay you until it gets your tax details. To help make the process less painful, it guides you to the correct form and lets you submit it online. However, Google won't prompt you for tax information until it has collected at least \$10 in advertising revenue.

Finally, fill in your name, address, and phone number.

Along the way, you need to tick the checkboxes next to several disclaimers, vouching that you won't click your own ads, place ads on pornographic sites, and so on.

Figure 14-3:
Google AdSense collects all the information it needs, from details about your website to your address, in a single page.

The screenshot shows the 'Website Information' section of the Google AdSense setup. It includes fields for Website URL (www.makingloadsofcash.com), Website language (English – English), and Contact Information (Account type: Individual, Country or territory: United States). A note states: 'Important - Your payment will be sent to the address below. Please complete all fields that apply to your address, such as a full name, full street name and house or apartment number, and accurate country, ZIP code, and city. [Example](#)'. Below this, Payee name (full name) is listed as Jonathan Hansai, with instructions: 'Your Payee name needs to match the name on your bank account.' and 'Payee must be at least 18 years of age to participate in AdSense.'. Street Address is listed as 2 Tupperware Lane, City/Town as Wachitou, and State as Alaska. At the bottom is a 'Done' button.

2. Once you finish, click Submit Information.

The next page summarizes all the information you supplied.

3. If you already have a Google account, choose “I have an email and password I already use with Google services.” Otherwise, choose “I do not use these other services.”

Google provides a dizzying number of online services. To prevent you from having to track dozens of passwords, it's a good idea to group them all under one Google account. If you already have a Google account, fill in your email address and password. If you don't, fill in your email address and choose a password.

4. Click Continue to finish the process.

Now you need to wait for Google to contact you by email to confirm your account. This involves two steps. First, Google sends you an email confirmation message almost immediately. This message contains a link you click to confirm your email address.

Then someone at Google needs to take a quick look at your site to confirm that it really exists and that it isn't promoting illegal activity (for example, offering

pirated copies of Windows 7). Once they do this, which usually takes a couple of days, you'll get a second message confirming that Google has activated your account.

UP TO SPEED

AdSense Rules

Google enforces a handful of rules that your website has to follow. Many are common sense, but it's still worth taking a quick look at them.

- You can't put the Google ads in email messages or pop-up windows; the temptation for spammers to abuse the system is just too great.
- You can't put ads on pages that don't feature any "real" content. This includes error, login, registration, welcome, and under-construction pages. You definitely can't create pages that include nothing but ads.
- You can't try to obscure parts of an ad (for example, by placing other elements over them using a style

sheet). The entire content of an ad needs to be visible.

- You can't click your own ads. You also can't use automated programs to do this for you. Finally, you can't entice your visitors to click your links using threats or incentives.
- Your website can't include excessive profanity, copyrighted material, pornography, content about hacking high-tech security systems, advocacy for illegal drugs, hate speech, or anything related to gambling.

To read the full AdSense policy, visit www.google.com/adsense/policies.

The AdSense Interface

Now that you have an AdSense account, you're ready to generate some ads and put them on your site. Go to www.google.com/adsense and log in with your email address and password. Google's AdSense page appears, divided into four sections by tabs at the top (see Figure 14-4).

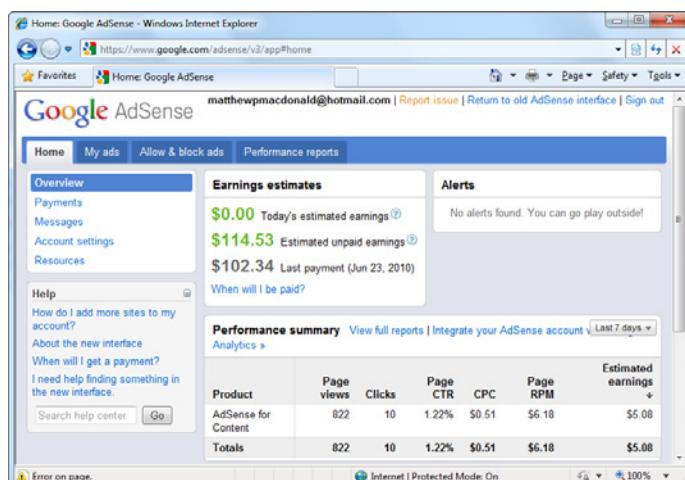


Figure 14-4:
Google divides its AdSense page into several tabs. Initially, you begin at the Home tab, where you configure your AdSense account and review the money you've made so far.

These sections include:

- **Home.** This tab gives you some critical summary information, including the amount of money you've made today, and notifications about any problems that prevent your ads from running (called *alerts*).
You'll also see some additional links in the panel on the left. For example, you can view a payment history that lists each check Google mailed you (click *Payments*), and you can update the information you supplied when you registered, such as your mailing address and tax information (click "Account settings"). Or, click *Resources* to browse the AdSense blog, chat with others in AdSense's help forum, and watch some inspiring success stories on video.
- **My ads.** This is your starting point for generating AdSense ads—it's where you specify your ad's display format and get the code to insert into your web pages.
- **Allow & block ads.** Here, you manage one of AdSense's most advanced features: filtering out the ads you don't want. For example, you can tell Google to refrain from showing ads in certain categories or from specific websites. The idea here is twofold: to bar your competition and to ensure that your visitors see ads relevant to them. (If the ads aren't useful, your visitors are less likely to click, and your earnings will plummet.)
- **Performance reports.** These reports help you assess the success of your ads. You start out with a graph that charts how much money you made over the last week. You can customize this graph by changing options and playing with its filters, until you find exactly the data you need. And if you want to do some heavy-duty analysis, you can download the numbers in an Excel-friendly CSV format in a single click.

Note: No matter which report you run, Google won't tell you what each individual click was worth or which particular ad caught a reader's eye. Instead, it gives you an estimate of the click value on a given day (what Google calls the CPC, or *cost-per-click*). Google also displays other useful information, like the percentage of times your visitors clicked your ads (called the CTR, or clickthrough rate). For example, a clickthrough rate of 2 percent means that if your page was requested 100 times, an ad on the page was clicked just twice.

Creating an Ad

Google provides four basic types of ad:

- **Text ads.** These are short, text-only pitches, like the ones you saw in Figure 14-2. They start with a link, followed by a snippet of text, and may include the advertiser's website name at the bottom. In total, each ad has just 100 characters. Text ads are shown in groups of one to four ads.

Note: Text ads are still the most popular type of Google ad. They're particularly adept at blending into the background, which means less irritation for your visitors (which is good) and fewer clicks (which is not so good).

- **Image ads.** These ads are actually images that range in size from banner strips to large squares. Image ads are more obtrusive than text ads, but they're steadily gaining in popularity. If your web page already has plenty of pictures, image ads look particularly good—for example, you'll often see them blending in at the side of a news article.
- **Rich media ads.** These ads show small Flash animations or videos. They use a polite model called click-to-play, which means the audio and video won't start until a visitor clicks the tiny play button (and he can control the volume with a nearby volume slider). If your guest doesn't click Play to start the video, the rich media ad looks just like an ordinary image ad.
- **Link unit.** This is a compressed box of links, with no descriptive text. The box has the title "Ads by Google" and the links are one- or two-word entries, like "Digital Cameras" or "Consumer Electronics." If a visitor clicks a link, Google serves up a new page filled with text ads for products in that category, and you get paid.

Before you can generate the right ad unit, you need to have a basic idea of where you plan to put your ads. Consider whether you want a vertical or horizontal strip of ads, and how wide or long that bar should be. Figure 14-5 previews just a few of your layout options.

When you're ready to dive in and build your first ad unit, follow these steps:

1. Click the "My ads" tab at the top of the page.

This opens the "My ads" section, which lists all the ads you've created. Right now, the list is empty.

2. Click the "New ad unit" button, which appears under the "Content > Ad units" heading.

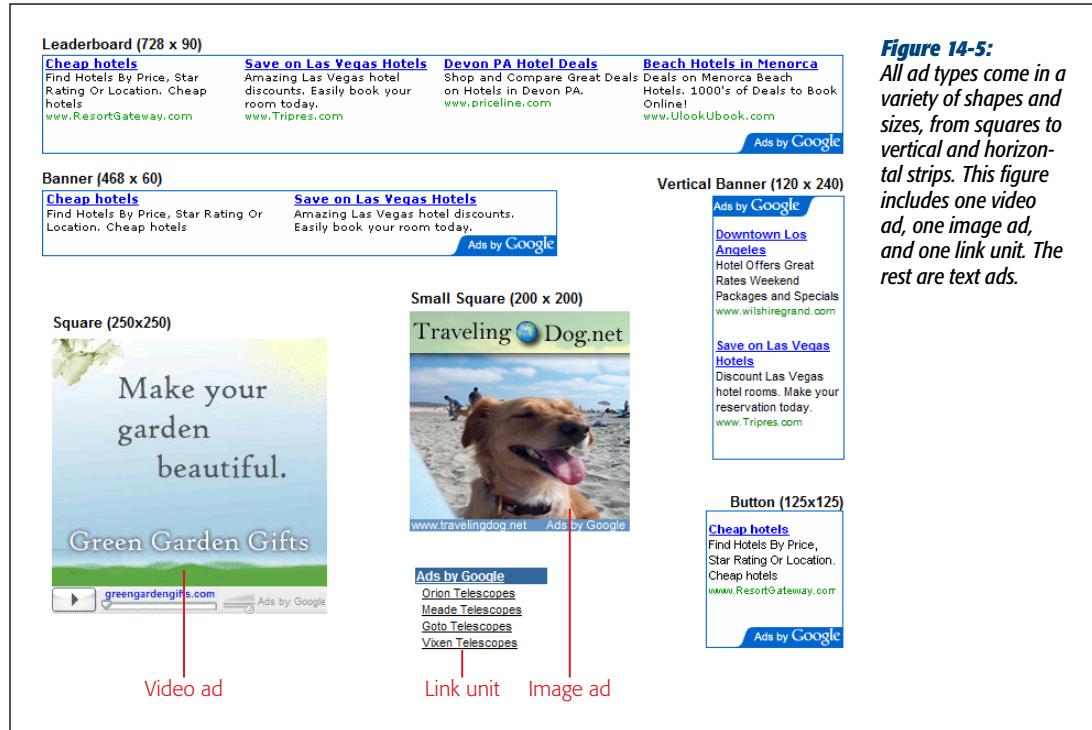
Now it's time to fill in the information for your ad.

3. Choose a name for your ad unit.

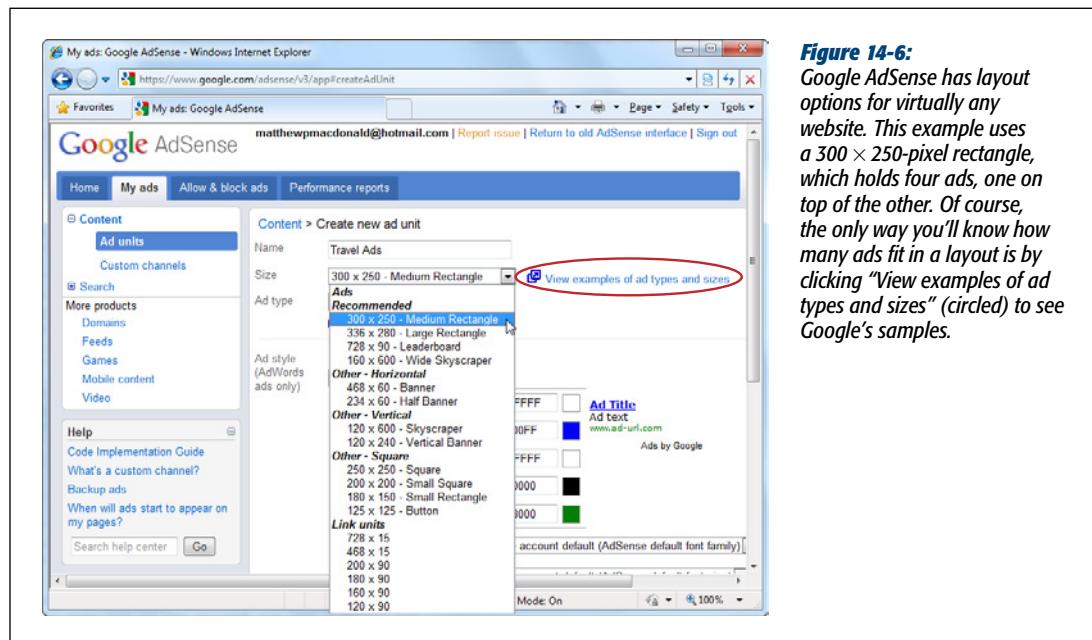
This name doesn't actually appear in the ad; it just makes your life easier as you manage your ads. After you create your ad unit, you can call it up by name (for example, Travel Page Ads) and modify it. This saves you the trouble of rebuilding your ad unit from scratch.

4. Choose the size of the ad box from the Size list (Figure 14-6).

Google always packages AdSense ads in boxes. A box can include one or several ads. The format you choose determines whether you'll get a vertical stack of ads or a horizontal row. It also determines how many ads you see at once (from one to five).

**Figure 14-5:**

All ad types come in a variety of shapes and sizes, from squares to vertical and horizontal strips. This figure includes one video ad, one image ad, and one link unit. The rest are text ads.

**Figure 14-6:**

Google AdSense has layout options for virtually any website. This example uses a 300 x 250-pixel rectangle, which holds four ads, one on top of the other. Of course, the only way you'll know how many ads fit in a layout is by clicking "View examples of ad types and sizes" (circled) to see Google's samples.

Tip: It's hard to picture different ad layouts, so Google makes it easy. Click the "View examples of ad types and sizes" link, and a window opens, displaying an example of every layout option. (Several of these examples turned up in Figure 14-5.) Using this page, you can find the format that best suits your site.

5. Choose a type of ad from the "Ad type" list.

Here's where you choose whether you want to stick to text ads only, allow images and rich media ads, or *require* images and rich media ads. Generally, image and rich media ads stand out more than text ads. However, you need to balance two conflicting goals: your desire to make money by attracting clicks with eye-catching ads and your desire to minimize distraction on your page by choosing less obtrusive ads.

6. Customize your ad's colors, borders, and fonts (optional).

The *color palette* sets the colors Google uses for the ad unit's text, background, and border (see Figure 14-7). In the "Color palette" list, you can choose from Google preset palettes, like Open Air and Blue Mix. Or, if you want to make sure your colors match the ones on your site, you can adjust them yourself. For example, you might want the ad box's border or background color to blend in with the background on your page. To change the colors of an ad unit, modify the color codes in the boxes underneath the list of color palettes.

Similarly, you can pick a font family (from a very limited list) and font size. You can also choose whether the corners of the box around your ad unit should be squared off or curved. (Or, if you don't want any border, just choose white for the border color.)

The screenshot shows the 'Customize Ad' interface for AdWords ads. On the left, there's a sidebar labeled 'Ad style (AdWords ads only)' with a dropdown menu set to 'Seaside'. The main area is titled 'Color palette' with a dropdown menu set to 'Seaside'. Below are color swatches and hex codes for Border (#6F3C1B), Title (#0000FF), Background (#FFFF66), Text (#000000), and URL (#008000). To the right of these is a preview window showing a yellow rectangular box with the text 'Ad Title' in blue, 'Ad text' in black, and a URL 'www.ad-url.com' in blue. At the bottom of the preview window, it says 'Ads by Google'. Below the preview are dropdown menus for 'Font family' (set to 'Use account default (AdSense default font family)'), 'Font size' (set to 'Use account default (AdSense default font size)'), and 'Corner Styles' (with three options: square, rounded, and sharp).

Figure 14-7:
As you change the colors, font, and borders around your ad unit, Google previews the results in a single-ad format.

Tip: Usually, you choose background and text colors that match the colors you already use on your web page. If you don't change the standard colors, your ad box will have the same background color as your web page (because it's transparent), and no border. For advice on how to choose custom HTML colors, see page 150.

7. Choose a channel (optional).

If you generate a half-dozen ads and scatter them on different pages throughout your site, you don't know which ones are making you money. Google's ad sales report shows you only the *total* number of clicks for all the pages on your site. Many site owners want more detail about which ads are working. Enter Google's *channels* feature.

To track the performance of different ads, you place each ad in a separate, virtual "channel." Google lets you create reports that compare each channel so you can tell which ads perform best.

To create a new channel, click the "Create new custom channel" link, which pops open a window asking you to name the channel.

Once you create a channel, you can apply it to the ad by clicking the tiny "add" link next to the channel name (Figure 14-8). (By the same token, click "remove" to stop tracking an ad.)

Tip: Channels are a great way to try out different ad strategies, and see which ad format and placement have the most success coaxing clicks out of your visitors. You can add multiple ads to the same channel to track them as a group, or you can create a separate channel for each ad.

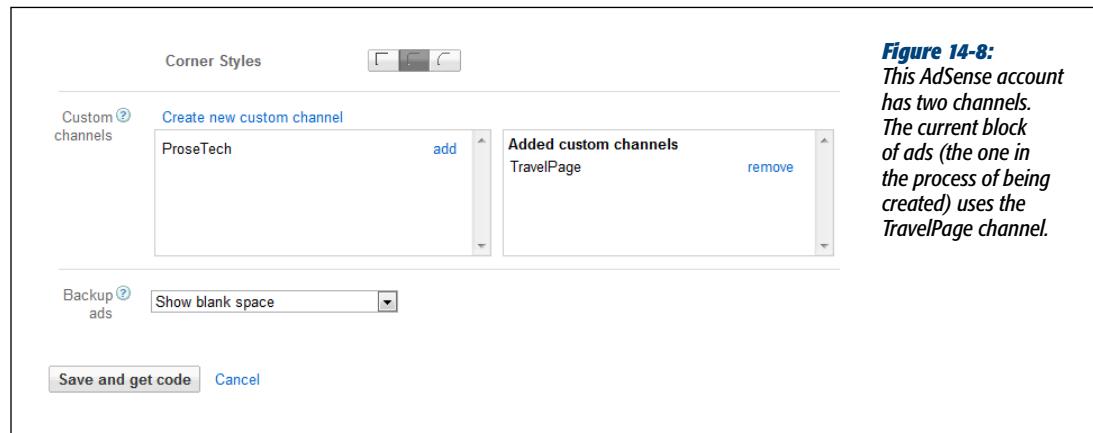


Figure 14-8:
This AdSense account has two channels. The current block of ads (the one in the process of being created) uses the TravelPage channel.

8. Choose a Backup ad.

When you first put an ad unit on a page, Google doesn't yet know what ads are a good match for your content, so it temporarily uses alternate content. Ordinarily, Google uses public service ads—messages from nonprofit organizations like Unicef, for example. Not long thereafter, Google's text-sniffing software pays a visit to your page, and the real ads materialize within a couple of days.

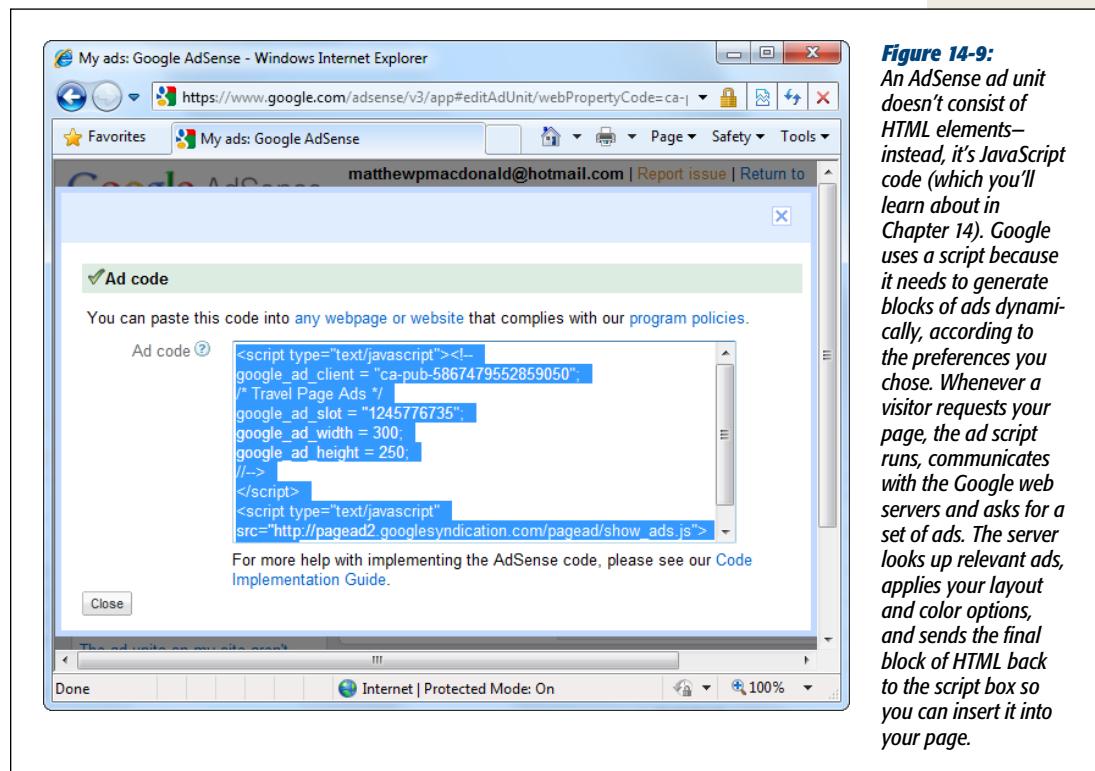
If you don't want to put up with generic ads, you can display alternate content.

You have two options—you can choose a solid color, in which case Google fills the ad unit with that color only—it doesn't include any ad content. The idea is that you use a color that matches the background of your page, so the “ad” disappears entirely. Your second option is to specify a URL for a page you want to use. Until the real ads are ready, that content appears on your page.

Tip: Alternate ads probably aren't worth the trouble. It's better to use the generic ads, because the ad layout is the same as the solid-color “ad,” which makes it easy to place the pitch in the right place and get an idea of what it looks like alongside the rest of your content.

9. Click “Save and get code.”

Once you do, a window pops up with the JavaScript code for your ad (see Figure 14-9).



10. Copy your ad code, and then click Close.

Click in the text box to select your ad unit code, right-click the selection, and then choose Copy. You're now ready to paste the code into one or more web pages, as described in the next section.

Note: If you need to modify your ad later, log back into Google AdSense, click the “My ads” tab, and find your ad in the list. When you click it, Google displays its details and lets you edit them. Most changes—say, altering the ad box color scheme—take effect immediately. However, if you want to change the format of your ad box, you need to get new code and paste it into your pages, because the size of your ad unit will change.

Placing Ads in Your Web Pages

After Google creates your ad script, you’re ready to pop it into your web page. Horizontal strips are the easiest to position; you simply paste the entire script right where you want it to appear.

Here’s an example that places ads at the bottom of a page:

```
<!DOCTYPE html>

<html>
<head>...</head>
<body>
    <h1>A Trip to Remember</h1>
    <p>
        After returning from my three-month travel adventure ...</p>
    <p>I hope you enjoy these pictures as much as I do.</p>
    <p>See pictures from ...</p>

    <script type="text/javascript"><!--
        google_ad_client = "ca-pub-5867479552859050";
        google_ad_slot = "1245776735";
        google_ad_width = 300;
        google_ad_height = 250;
    //-->
    </script>
    <script type="text/javascript"
        src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
    </script>
</body>
</html>
```

Figure 14-10 shows the result.

Positioning vertical ads requires a little more work, but it’s easy once you learn the trick. The challenge is to flow the rest of your page content *beside* the vertical ad. As you learned in Chapter 9, you can use style sheet rules to float the ads on the side of a page.

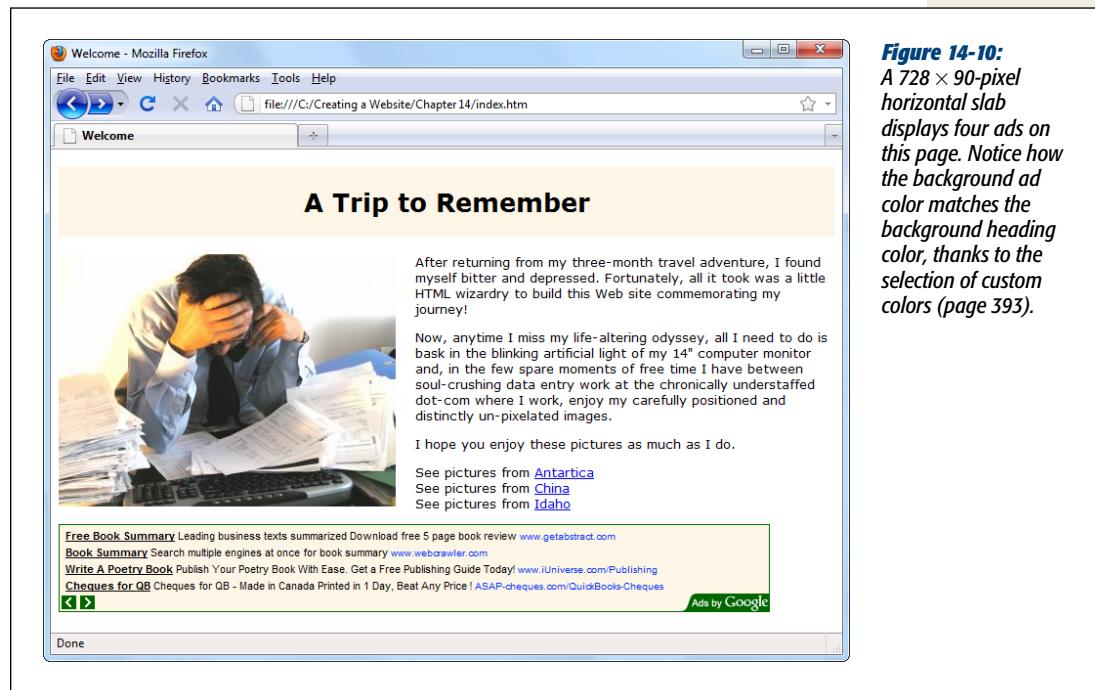


Figure 14-10:
A 728 × 90-pixel horizontal slab displays four ads on this page. Notice how the background ad color matches the background heading color, thanks to the selection of custom colors (page 393).

To use the style sheet approach, begin by wrapping your ad script in a `<div>` element. Here's an example featuring the content you saw in Figure 14-10:

```
<!DOCTYPE html>

<html>
<head>...</head>
<body>
  <div class="floatRight">
    <h1>A Trip to Remember</h1>

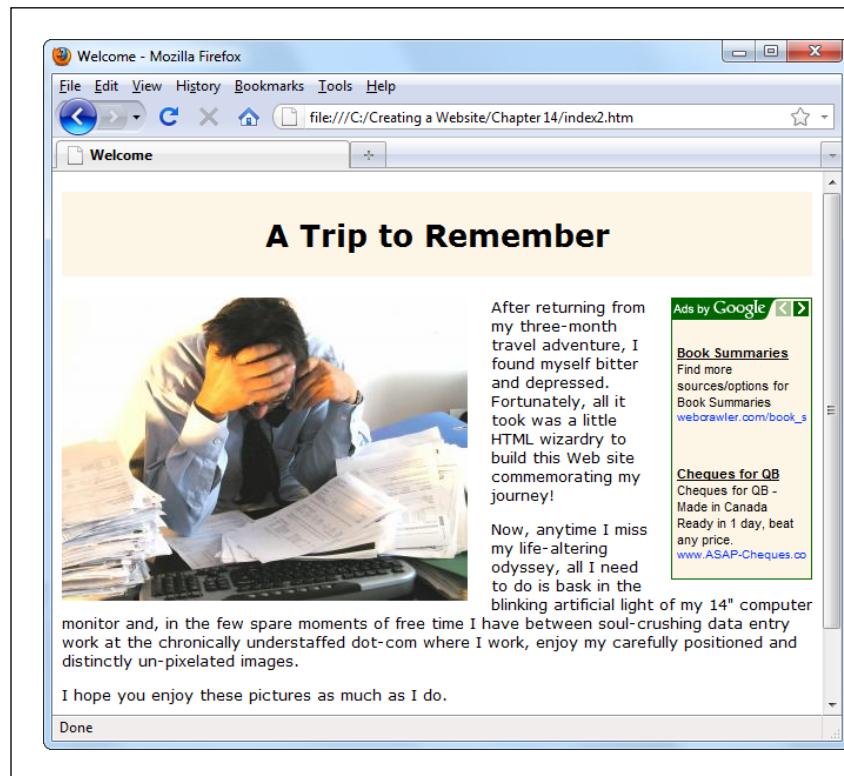
    <script type="text/javascript"><!--
      google_ad_client = "pub-5876479552859050";
      google_ad_slot = "4493177655";
      google_ad_width = 120;
      google_ad_height = 240;
    //-->
    </script>
    <script type="text/javascript"
      src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
    </script>
  </div>
```

```
<p>
    After returning from my three-month travel adventure ...</p>
<p>I hope you enjoy these pictures as much as I do.</p>
<p>See pictures from ...</p>
</body>
</html>
```

Notice that the `<div>` element (which has no formatting on its own), uses the style sheet class `floatRight`. In your style sheet, you use the rule below to make the `<div>` section float using the `float` attribute (see page 196):

```
.floatRight {
    float: right;
    margin-left: 20px;
}
```

Figure 14-11 shows the result.



The screenshot shows a Mozilla Firefox window displaying a web page titled "A Trip to Remember". The page content includes a photograph of a person sitting at a desk overwhelmed by stacks of papers, followed by a paragraph of text. Below the text is a horizontal line and the phrase "I hope you enjoy these pictures as much as I do.". To the right of the main content, there is a vertical banner ad titled "Ads by Google". The banner contains two ads: one for "Book Summaries" and another for "Cheques for QB". The "Book Summaries" ad includes a link to webrrawler.com/book_s. The "Cheques for QB" ad includes a link to www.ASAP-Cheques.co.

Figure 14-11:
A 120 × 240-pixel vertical banner fits two ads on this page. Vertical ads are the most popular format for Google ads. Not only do they tuck in neatly next to a web page's content, but if you make them long enough, they remain visible even as your visitors scroll down the page.

UP TO SPEED

How AdSense Creates Targeted Ads

Every time you serve up a web page that contains Google ads, the AdSense script sends a message to Google's web server asking for ads. This message includes your ad preference information and your unique client ID. (Your client ID is something like *pub-5876479552359052*; you can see it in the script code.)

The first time Google receives this request, it realizes that it hasn't examined your page yet, and it doesn't know what types of ads are best suited for it. So it sends you a block of generic ads (or your alternate content, if you chose that feature, as described on page 394). Google also adds your page to a list of pages it needs to visit. Sometime in the next couple of days, the Google bot heads over to your site and analyzes its content. From that point forward, you'll see ads based on the content of your page.

If 48 hours pass and you still aren't getting targeted ads, there could be a problem. One of the most common mistakes is putting ads on pages that don't have much text, in which case Google can't figure out what your site is re-

ally all about. Remember, Google AdSense only considers a single page—the one with the ad unit—when it checks out your site. You can run into another potential problem if you put your ad on an inaccessible page. For example, the Google bot can't get to any page that's not on the Internet—pages on your personal computer or a local network just won't cut it. Likewise with password-protected pages. Some websites block robots through exclusion rules (see page 297), which stops the Google bot cold.

Finally, remember that Google may create ads that aren't appropriate for your site. For example, you might be discussing the pros and cons of a programming language called Python, and Google might respond with a promotional ad from a pet store. Often, Google will figure out this sort of problem on its own, both by analyzing your pages and by discarding ads that don't generate many clicks. But you may be able to help it rule out some inappropriate options using the ad-blocking feature. To get started, log into AdSense and click the "Allow & block ads" tab.

Google-Powered Searches

Google gives you another way to please your visitors (and earn some cash in the process). You can add a search box to your pages, letting visitors launch Google queries right from your site. Even better, you get the earnings for any ads they click in the search results—a feature Google calls (rather unimaginatively) AdSense for Search.

From your AdSense account, you can easily add a Google search box to your site:

1. **Log in to your AdSense account, and click the "My ads" link.**

This is the same place you create ad units, but you want to go to a different subsection to build your search box (Figure 14-12).

2. **In the panel on the left, click Search to see its subcategories. Click the "Custom search engines" link, and then click the "New custom search engine" button.**

Now you can fill in the information for your search box.

3. **Name the search.**

The name lets you retrieve your customized search box, tweak your settings, and get new code, without starting from scratch.

Use this section to create search boxes

Use this section to create ad units

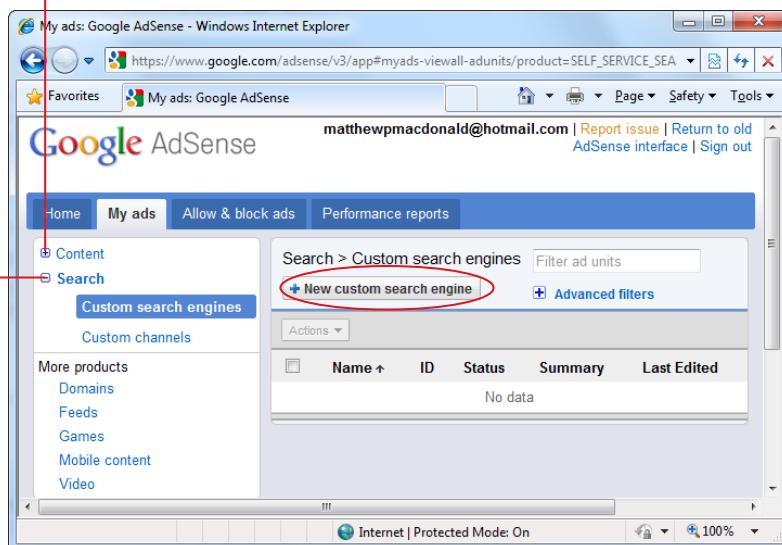


Figure 14-12:
So far, this AdSense member hasn't developed any search boxes. Clicking the "New custom search engine" button creates the first one.

4. Choose the search type.

Choose “The entire Web” to create a search box that uses the familiar Google search engine we all know and love.

Choose “Only sites I select” to restrict the search to a limited set of sites. You can use this feature to limit searches to your site only.

5. If you choose to limit the search, fill in a list of searchable sites in the “Selected sites” box (Figure 14-13).

You can enter individual pages (as in <http://www.ChocolateMysteries.com/recipe01.html>) or use an asterisk to represent all the pages in a folder or on a site (as in http://www.ChocolateMysteries.com/*), which is more common. If you need to enter multiple URLs, put them on separate lines in the box.

Note: Even when you limit searches to your website, Google polls its standard, centralized catalog of web pages—it just limits the results it displays to the pages from your site. If Google doesn’t have your pages in its catalog (either because you just created the site or because Google doesn’t know your site exists), these pages won’t turn up in any searches, no matter how you customize your search box. For a refresher about getting Google to notice you, see page 294.

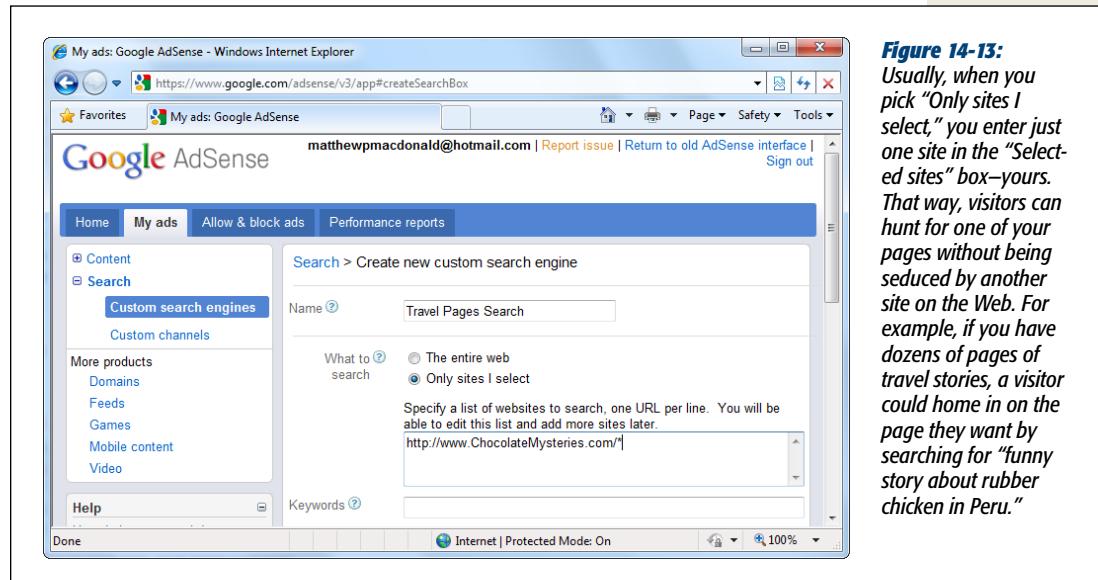


Figure 14-13:
Usually, when you pick “Only sites I select,” you enter just one site in the “Selected sites” box—yours. That way, visitors can hunt for one of your pages without being seduced by another site on the Web. For example, if you have dozens of pages of travel stories, a visitor could home in on the page they want by searching for “funny story about rubber chicken in Peru.”

6. Fill in the Keywords box (optional).

Google automatically adds any keywords you include here to your visitors’ searches. This gives you a way to design a search box that’s targeted to certain types of content. For example, if your site is all about golf, you might include the keyword *golf*. That way, if a visitor searches for *tiger*, the search returns pages about Tiger Woods, not the African savannah.

7. Check the SafeSearch box, if you want to switch it on.

SafeSearch filters profanity and sexual content from search results. You’ll find SafeSearch useful in two situations: First, it’s de rigueur for sites that provide children’s content. Second, it’s handy if your website deals in a topic that shares some keywords with adult-only sites. For example, if you create a breast cancer awareness page, you don’t want someone to type “breast exam” into your Web search box and dig up the wrong goods.

8. Tweak the country and language settings on the page, if they apply to you.

These settings include your website’s language and geographic location. As you probably know, Google has country-specific pages that can tweak search results, providing them in different languages or giving priority to local sites.

9. Optionally, click the plus (+) box next to the “Custom channels” section, and pick a custom channel.

You can place your search box in a specific channel, just as you can an ad unit, which is useful if you want to track the ad dollars you make from the search box. See page 394 for more information about channels.

10. Click the plus (+) box next to the “Search box look and feel” section, so you can tailor the appearance of the Google search box (optional).

There's not a lot to change here. You can alter the size of the search box and the placement of Google's logo. You can also choose between a white, black, and dark gray background.

11. Optionally, click the plus (+) box next to the “Search results and ad location” section, so you can customize how Google displays its search results.

Using these settings, you can set the colors Google uses in its search results. This way, the results can blend in with the color scheme on the rest of your site. This feature is almost the same as the color palettes for AdSense ad units (page 393).

You can also choose where to display the search results:

- Choose “on a Google page in the same window” to replace the current page with Google's standard search results page.
- Choose “on a Google page in a new window” to pop open a new browser with the search results. Pop-up windows are usually annoying to web visitors, but this technique is handy if you want to make sure your website sticks around on your visitor's desktop.
- Choose “on my website using an iframe” to keep your visitors on your website, and show the search results alongside your content. This is everyone's favorite option, but it requires slightly more work because you need to create two pages: one with the search box, and one that holds the search results. Google gives you some markup to place on each page.

If you want to use this option, you need to supply the URL for your search results page (for example, <http://www.ChocolateMysteries.com/searchresults.html>). Don't worry if you haven't created this page yet—you can create and upload it when you finish creating the search box. You also need to tell Google how wide the search results should be. The standard option is 800 pixels, which is a good choice if you don't plan to pad the sides of your page with extra content (like ads or a menu bar). If you have other content you want to show on the page, you can place it above the search results.

12. Check “I have read and agree to the Google Custom Search Terms of Service,” and then click the “Save and get code” button.

The final setup page includes the markup for your complete, customized search box (see Figure 14-14) in a `<form>` element. (Page 453 has more about forms in HTML.) As with the AdSense code, you can paste this HTML into any web page.

If you chose to use a search results page on your own site, you'll get a second box with the markup you need to create that page. This consists of a `<div>` element and some JavaScript code. It works in much the same way as the AdSense code. You simply place the `<div>` element where you want the search results to appear.

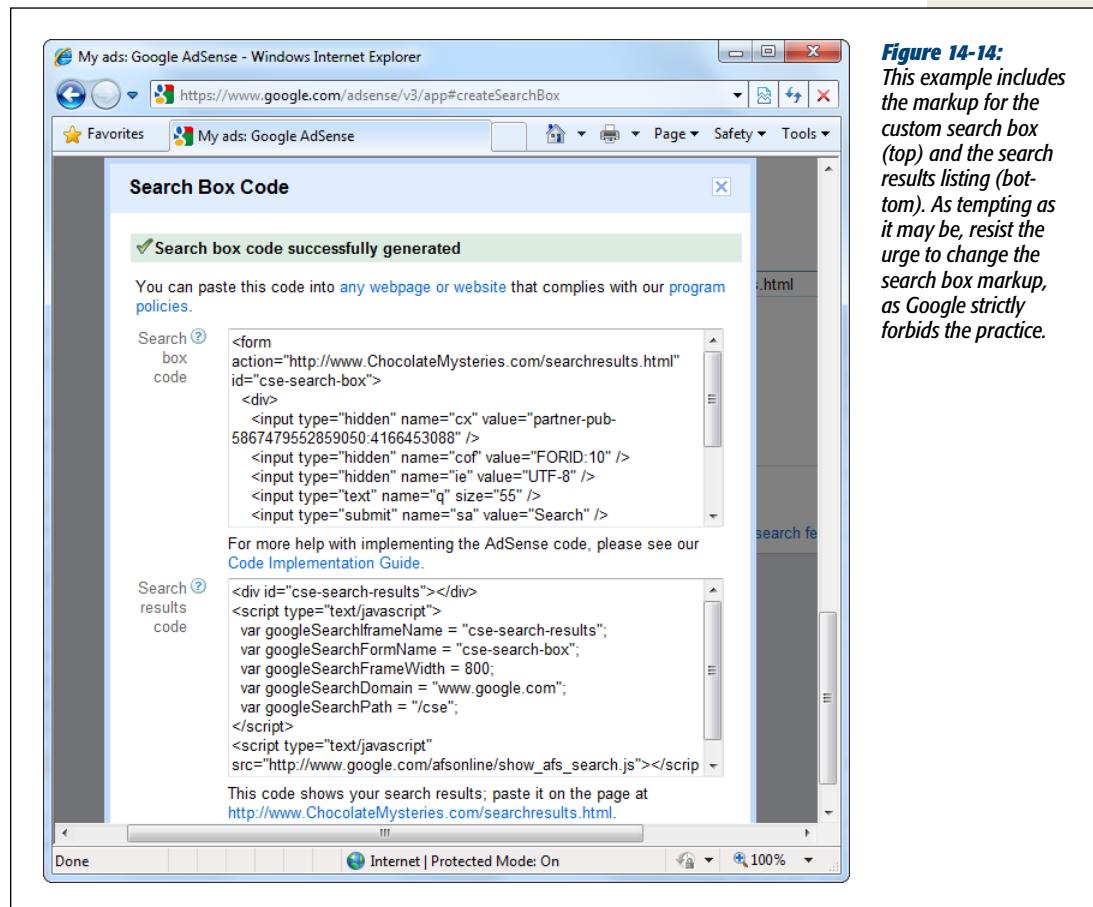


Figure 14-14:
This example includes the markup for the custom search box (top) and the search results listing (bottom). As tempting as it may be, resist the urge to change the search box markup, as Google strictly forbids the practice.

Amazon Associates

As popular as ads are with website owners, they have one serious drawback: They clutter up your pages. Once you perfect a design with carefully chosen pictures and style sheets, you might not want to insert someone else's ad. And although Google ads aren't as visually distracting as other ads can be, like animated banners or pop-up windows, they still chew up valuable screen space. If you can't bear to disturb your web page masterpieces, you might be interested in a subtler option.

Amazon Associates is the Web's longest-running affiliate program. If you have a personal site with a "favorite books" page, or if you just refer to the odd book here and there, you might be able to make some extra money by signing up as an affiliate.

The basic idea behind the Amazon Associates program is that you provide links to book and other product pages on Amazon's website. For example, if you write a

blurb about a great recipe, you could add a link to the Amazon page that sells the cookbook you're quoting. The link itself is a nice feature for your site, since it provides visitors with more relevant information. But the best part is what happens if a visitor decides to buy the book. You wind up making a healthy commission of 4 percent of the book's sale price.

Note: Amazon commissions aren't just for books. You can provide links to pretty much everything for sale on Amazon (except items that other retailers, like Target and Office Depot, sell). But AdSense limits how much you can make on nonbook items. For example, with personal computers, you're capped at a maximum \$25 commission per item. You can also earn more than 4 percent for some category types (like MP3 downloads) or if you sell a certain number of items per month. These rules change from time to time, so make sure you scour the Amazon Associates website carefully to get the lowdown.

Signing Up As an Associate

Signing up for the Amazon Associates program is even easier than joining AdSense. Just follow these steps:

1. Go to <http://affiliate-program.amazon.com>, click the “Join now” button, and then log in with your Amazon email and password.

To join the associates program, you need an Amazon account. If you don't have one, click “I am a new customer” to create an account.

2. Enter your personal information and your website information.

For your personal information, you need to supply your name, address, and telephone number. For your website information, you need to supply a website name, URL, and brief description (see Figure 14-15).

3. Click Finish to submit your application.

Shortly afterward, you'll get a confirmation message saying that you're approved on a trial basis. This email includes your unique associate ID. This number is important, because it's the single piece of information you need to add to all your Amazon links to start earning commissions. You can now use the associate tools at <http://associates.amazon.com> (see the next section).

In a couple of days, when someone at Amazon verifies your site and confirms that it doesn't run afoul of the law, you'll get a second message confirming that you're in for good.

4. If you'd like to tell Amazon how to pay you right now, click Specify Payment Method Now.

You can choose your preferred form of payment even before Amazon officially accepts you into its program. Your choices include payment by check, Amazon gift certificate, or direct deposit to a U.S. bank account. Amazon doesn't send checks until you make at least \$100, and it charges you a \$15 processing fee. Other payment types kick in once you reach \$10, and they don't involve any fees.

The screenshot shows a Mozilla Firefox browser window with the title bar "Amazon.com Associates Central - Mozilla Firefox". The address bar shows the URL "https://affiliate-program.amazon.com/gp/assoc". The main content area is titled "Website Subject & Content". It contains several input fields and descriptions:

- What is the name of your website?** The input field contains "The Cynical Reviewer". A note below says, "We will use this name to generate your unique Associates ID."
- What is the URL of the website(s) you will use to send traffic to Amazon?** The input field contains "http://www.thecynicalreviewer.com". A note below says, "Your website, your blog, your Twitter feed, etc." There is also a link "(Add additional URLs*)".
- What is your website(s) about?** The input field contains "A website that gives profoundly depressed reviews of romantic comedies and other films." A note below says, "What can users do on your website, who is it for, and what kind of products do you intend to promote?"
- Which of the following topics best describes the topic of your website(s)?** A dropdown menu is open, showing "Movies/DVDs/TV".

At the bottom right of the form is a "Done" button.

Figure 14-15:
To become an Amazon associate, you need to supply some basic information about your site. Don't skip this step, because someone from Amazon will take a quick look at your site before she approves you for the program.

Generating Associate Links

Once you have your associate ID, you can create *associate links*, the hyperlinks that send your visitors to Amazon. The trick is formatting the URL in the right way.

You add your associate ID to the very end of the associate link. For example, the first email Amazon sends includes an example of the associate link to its home page. It looks like this:

```
http://www.amazon.com?tag=prosetech-22
```

In this example, the associate ID is *prosetech-22*. (Replace it with your own ID to create a link for your website.) If someone follows this link and buys something, you earn a 4 percent commission.

Here's how you link to the Amazon page using an anchor element:

Visit `Amazon` and help me save up to buy a Ferrari.

Tip: Amazon encourages you to advertise the fact that you're an associate. If you'd like to boast, the company provides a collection of ready-made Amazon logos and banners at <http://tinyurl.com/46cq84>. You can add these to your site, and even put them in anchor elements to transform them into associate links. (You have to be an Amazon associate to view the banner page.)

Product links

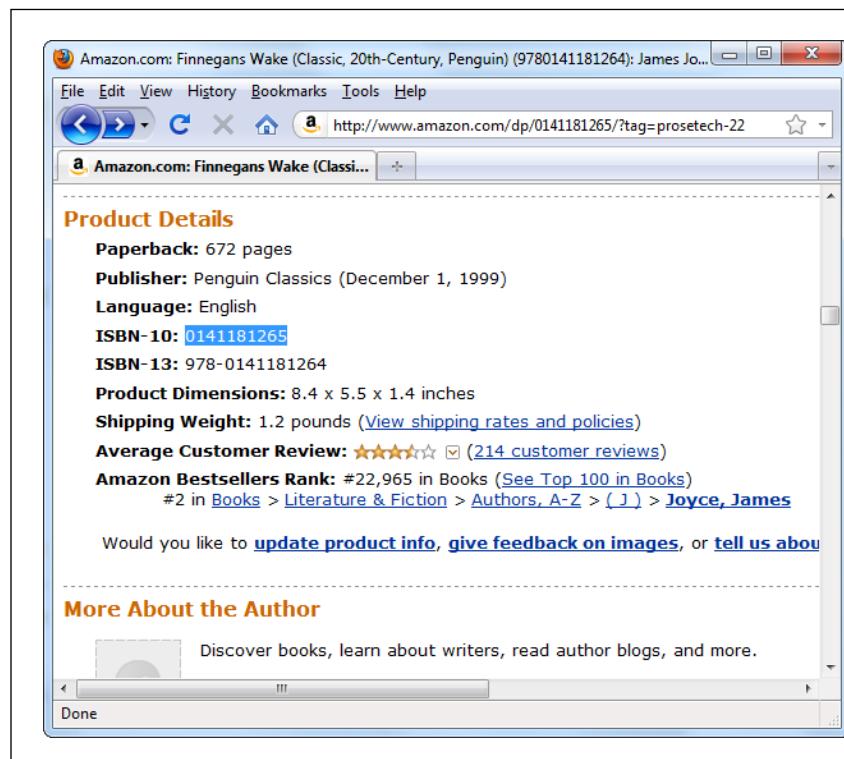
You get better commissions with links that lead directly to a specific product. Amazon offers several associate link formats, and here's one of the simplest:

<http://www.amazon.com/dp/ASIN/?tag=AssociateID>

And here's a specific example:

<http://www.amazon.com/dp/0141181265/?tag=prosetech-22>

In this link, you customized two details, the ASIN (Amazon Standard Item Number) and the associate ID. The ASIN is 0141181265 (which leads to the book *Finnegans Wake*) and the associate ID is *prosetech-22*. Figure 14-16 shows you where to find an ASIN.



The screenshot shows a web browser displaying the product page for "Finnegans Wake (Classic, 20th-Century, Penguin) (9780141181264); James Jo...". The URL in the address bar is <http://www.amazon.com/dp/0141181265/?tag=prosetech-22>.

Product Details

- Paperback:** 672 pages
- Publisher:** Penguin Classics (December 1, 1999)
- Language:** English
- ISBN-10:** **0141181265**
- ISBN-13:** 978-0141181264
- Product Dimensions:** 8.4 x 5.5 x 1.4 inches
- Shipping Weight:** 1.2 pounds ([View shipping rates and policies](#))
- Average Customer Review:** ★★★★☆ (214 customer reviews)
- Amazon Bestsellers Rank:** #22,965 in Books ([See Top 100 in Books](#))
#2 in [Books](#) > [Literature & Fiction](#) > [Authors, A-Z](#) > ([J](#)) > [Joyce, James](#)

Would you like to [update product info](#), [give feedback on images](#), or [tell us about this item](#)?

More About the Author

Discover books, learn about writers, read author blogs, and more.

Done

Figure 14-16:
Every item in the Amazon catalog has a unique ASIN, which you can find in the Product Details section of that product's page. For books, the ASIN is the same as the 10-digit ISBN (highlighted), which is an industry-standard book ID number.

Here's an example of a complete link:

The development of the modern personal computer was first presaged in Joyce's [Finnegans Wake](http://www.amazon.com/dp/0141181265/?tag=prosetech-22).

That's all you need.

Note: If a website visitor follows a link to a specific Amazon product but goes on to buy something completely different, it's all good—you still get the same 4 percent commission.

Advanced links

Amazon has a set of specialized tools that help you generate links. Using these tools, you can create a range of snazzy links. Your options include:

- Links with thumbnail pictures
- Links to product categories (like *equestrian magazines* or *bestselling kitchen gadgets*)
- Ad banners that advertise a specific Amazon department
- Amazon search boxes that let visitors perform their own queries

Even if you don't want these fancier links (and if your life isn't dedicated to selling books, you probably don't), there's still good reason to build links with the tools Amazon provides: Their links have built-in tracking, so you can determine how many people see each link.

Note: Amazon tracking is very clever. Essentially, it embeds a tiny one-pixel image alongside each link. If someone requests a page that contains one of these links, the browser automatically fetches the invisible picture from Amazon. When Amazon gets the request for the invisible picture, it knows someone clicked the link, and it records a single *impression* (page view) in its tracking database.

Here's how you use Amazon's link-building tools:

1. Go to <http://affiliate-program.amazon.com>, and log in.

This takes you to the Associates Central home page, which offers a variety of reports for checking your sales progress to date, as well as tools for building links.

Tip: For detailed information about the more ambitious things you can do with Amazon Associates, click the Get Started Now button. You can also get invaluable advice from other associates by visiting the discussion forums—look for the Discussion Boards link at the bottom of the menu bar on the left.

2. Click the "Links & Banners" tab. Then, in the menu on the left, choose Product Links.

Amazon lets you build many different types of link. Product links point to individual items on Amazon's site. They're generally the most useful. But if you plan to go Amazon-crazy, feel free to explore all the other types of links.

3. In the search box, type the ASIN for your product, and then click Go.

If you don't know the ASIN, select what you think is the most appropriate category, and then type in the product name. When you click Go, Amazon searches for the product and lists the results (see Figure 14-17).

The screenshot shows the 'Product Links' tool on the Amazon Associates dashboard. The search bar at the top contains the text 'Hunting of the Snark'. Below the search bar, there are dropdown menus for 'Search' (set to 'All Programs') and 'within' (set to 'All Products'). A 'for' field contains the text 'Hunting of the Snark keyword or ASIN/ISBN'. A 'Get' button is visible next to the search input. Below the search area, the text 'Results for "Hunting of the Snark" from All Programs' is displayed. Underneath this, a section titled 'Results From Amazon (See all 332 results)' shows a single result: 'The Hunting of the Snark (Graphic Novel)' by Lewis Carroll, priced at \$10.17. To the right of the product title is a 'Get Link' button. The overall interface is a standard web-based form with a light blue header and white background.

Figure 14-17:

When you build a link, you can search for the specific product you want. This search works in more or less the same way as a search from the Amazon home page.

4. Click the Get Link button next to the product you want to link to.

You'll see a page that shows you the product and previews the link you're about to create.

5. Choose the link type, and customize its appearance.

You can configure your product link to be as fancy or as simple as you want (see Figure 14-18). To create a text link, choose Text Only as the link type. Otherwise, you'll get a more detailed box that includes a product picture and price.

You can choose whether your page opens the product-page link in a new browser window, how big the image is, what price information the product box includes, and what colors it uses.

Once you finish, copy the HTML from Amazon's text box and paste it in any web page on your site.

1. Select Link Type	
<input checked="" type="radio"/> Text and Image (Enhanced Display) <input type="radio"/> Image Only (Basic Display) <input type="radio"/> Text Only (Basic Display)	
2. Customize Link	
<input checked="" type="checkbox"/> Open link in new window <input checked="" type="checkbox"/> Show border <input checked="" type="checkbox"/> Use larger image	
Price Options: <input type="button" value="Show All Prices"/>	
Background Color:	<input type="text" value="FFFFFF"/>
Text Color:	<input type="text" value="000000"/>
Link Color:	<input type="text" value="0000FF"/>

Live Preview

This link is served by us. You do not need to download the graphic. Just highlight and copy the HTML code to the left, then paste it into the code for your Web site.



[The Annotated Hunting of the Snark](#)
Lewis Carroll, Hen...
Best Price \$9.50
or Buy New \$18.45
[Buy amazon.com from](#)
[Privacy Information](#)

1. Select Link Type	
<input type="radio"/> Text and Image (Enhanced Display) <input type="radio"/> Image Only (Basic Display) <input checked="" type="radio"/> Text Only (Basic Display)	
2. Customize Link	
Link Text: <input type="text" value="The Annotated Hunting of the Snark (The Annot..."/>	
Link Destination: <input type="button" value="Detail Page"/>	

Preview

To create this link, cut and paste the HTML code in the lower left textbox into your web page.

[The Annotated Hunting of the Snark](#)
([The Annotated Books](#))

When you create a text link, Amazon generates an anchor element that looks fairly complex. (As described earlier, the anchor element contains an invisible `` element that lets Amazon track how many times it displays the link.)

However, like all anchor elements, it's relatively easy to put this element where you want it. Just pop it into an existing paragraph, like this:

```
<p>Lewis Carroll's work as a mathematician may have driven him insane,  
as his famous book  
<a href="http://www.amazon.com/exec/obidos/...">The Hunting of the Snark</a>  
  
attests.</p>
```

Note: Amazon puts the full title of the book inside the anchor element. This might make your link a little longer than you intend, because it might include information about the edition or a byline. If so, just cut it down to the title you want to use.

Amazon sends you monthly emails to let you know how much you're earning, but if you can't stand the suspense, you can log in to Amazon Associates any time. Click the Reports tab (Figure 14-19) to get detailed information on how much you're earning per day, week, month, or quarter.

Figure 14-18:
As you choose your link options, Amazon previews the final result. You can choose to make a text-and-image link (top) or a plainer, easier-to-integrate text link (bottom).

The screenshot shows the 'Orders Report' section of the Amazon Associates dashboard. At the top, there are date selection dropdowns for 'Pre-selected Period' (set to 'Yesterday') and 'Exact Period' (set to 'From Feb 3, 2006 To Feb 3, 2011'). Below these are buttons for 'Display on page', 'Download Report (TSV)', and 'Download Report (XML)'. A note below the buttons states: 'Downloads include expanded ASIN, date, and other information.' The main area is titled 'Orders Report' and shows a summary for the period 'February 3, 2006 to February 3, 2011'. A 'Summary' link is available. The table lists items with orders, including their names, product link conversion rates, product link clicks, items ordered through product links, all other items ordered, and total items ordered. The items listed are:

Item Name	Product Link Conversion	Product Link Clicks	Items Ordered Through Product Links	All Other Items Ordered	Total Items Ordered
SPECIAL OFFER Executive 2-Button Italian Velvet Corduroy Blazer Sportcoat Extra Soft Super 140'S Fabric CHARCOAL (44 Regular)	N/A	0	0	1	1
Books					
A Little Princess (Unabridged Classics)	N/A	0	0	1	1
ASP .NET 2.0 Website Programming: Problem - Design - Solution (Programmer to Programmer)	N/A	0	0	1	1
Access 2007: The Missing Manual	N/A	0	0	2	2
Airbus A380: Superjumbo of the 21st Century	N/A	0	0	1	1

Figure 14-19:

Amazon provides many different types of reports. To get a fascinating look at the items your visitors are buying, click *Orders Report* in the menu on the left.

DESIGN TIME

Sell Your Custom Designs

It's no secret that the Web is full of small-scale businesses selling handmade and custom goods. You can expand your product offerings beyond those from Amazon, but it might take some effort. Depending on what you want to sell, you might need to search out the right wholesalers or craftspeople, or just build a woodworking shop in your basement. But if you have something a bit simpler in mind, like customized clothes, mugs, hats, magnets, posters, buttons, iPhone cases, or yoga mats, then CafePress (www.cafepress.com) and Zazzle (www.zazzle.com) are your new best friends.

Both sites let you design your own goods and offer them for sale on a personalized section of the website (no HTML coding required). The genius of CafePress and Zazzle is that they let you use your own artwork. If, for example, you want to create a custom beer mug, you aren't limited to typing in a cheesy message and picking a preset font. Instead, you can upload your own graphics file (whether it's a digital picture or Photoshop masterpiece), and slap that on the side of your product. Of course, life isn't perfect.

On many products, CafePress and Zazzle limit where you can place your pictures. For example, you can put graphics on the front and back of most clothes, but you can't fill the whole surface or wrap around the sides. (Zazzle's shoes are an eye-catching exception. They let you wrap graphics along the entire outer surface.)

Once you create a design, you can order one for yourself, or peddle it to an audience of millions by sending your visitors to an automatically generated store page on the CafePress or Zazzle site. If someone buys an item, you get a small cut of the price, and CafePress or Zazzle processes the payment, prints the item, and ships it to your customer.

If you're interested in the idea but still in search of some inspiration, head to Zazzle and click the Search button without typing in a product name. You'll see a list of everything they offer, organized so that the bestsellers come first. Timely t-shirts with corny graphics, sophomoric slogans, or a reference to politics or current events are always among the most popular.

PayPal Merchant Tools

Unless your website is wildly popular, ads and other affiliate programs will net you only spare change. If you have all-consuming dreams of web riches, you need to actually sell something.

You don't need to go far to run into self-made Internet commerce kingpins. A surprisingly large number of people have made their living with creative products. Examples include t-shirts with political catchphrases, empty bottles of wine with *R.M.S. Titanic* labels, and collectable toys from a relative's basement. Your path to a thriving e-business might involve little more than buying tin spoons from Honest Ed's and decorating them with macramé.

No matter how good your goods are, you need a way to sell vast quantities of them easily and conveniently. Very few people will go through the hassle of mailing you a personal check. But if they can make an impulse purchase with a credit card, your odds of making a sale improve significantly.

Accepting credit cards isn't the easiest thing in the world to do. You can do so in two ways:

- **Open a merchant account with a bank.** This is the traditional way businesses accept credit cards. Requirements vary from country to country, but you may need a business plan, an accountant, and some up-front capital.
- **Use a third-party service.** A number of companies accept credit card payments on your behalf in exchange for a flat fee or a percentage of the sale. In this chapter, you'll learn how to use one of the best—PayPal.

Unless you have a large business, the second option is always better because of the additional risks that accompany web-based sales.

First of all, the Internet is an open place. Even if you have a merchant account, you need a *secure* way to accept credit card information from your customers. That means the credit card number needs to be *encrypted* (scrambled using a secret key) so that Internet eavesdroppers can't get at it. Most webmasters don't have a secure server sitting in their basement, and many web hosts charge extra for the privilege of using theirs.

Another problem is that when you conduct a sale over the Web, you don't have any way to collect a signature from the e-shopper. This makes you vulnerable to *charge-backs* (see the box on page 412).

Note: PayPal is a staggeringly large Internet company that offers its payment services in well over 100 countries. If you were to rank banks by the sheer number of accounts they hold, PayPal (with more 200 million account members) would be the largest bank in the world. The company was established in 1998 and purchased by eBay in 2002.

FREQUENTLY ASKED QUESTION

Understanding Chargebacks

What's a chargeback?

Credit card companies issue a chargeback when a cardholder asks them to remove a charge from their account. The buyer may claim that the seller didn't live up to their end of the agreement or that he never made the purchase in the first place. A chargeback can occur weeks or months after someone buys an item.

From the buyer's point of view, a chargeback is relatively easy. He simply phones the credit card company and reverses the transaction. The money you made is deducted from your account, even though you already shipped the product. If you want to dispute the buyer's claim, you're in the unenviable position of trying to convince a monolithic credit card company to take your side. Many small businesses don't dispute chargebacks at all, because the process is too difficult, expensive, and unsuccessful.

However, when you use a third-party service, the odds tilt in your favor. If the buyer asks for a chargeback, the charge-

back is made against the third-party company that accepted the payment (like PayPal), not you. And even though PayPal isn't as large as the average multinational bank, it's still a major customer of most credit card companies, which means it has significant clout to fight a chargeback.

The end result is that buyers are less likely to charge back items when they pay through PayPal. And if they do, PayPal gives you the chance to dispute the chargeback. It even lets you contact the buyer to see if there's a simple misunderstanding (for example, to check whether you sent the item to the wrong address). And if you're really paranoid, you can use PayPal's Seller Protection Policy which, if you take a few additional steps (like retaining proof of delivery), insures you for up to a \$5,000 loss. For more information about how PayPal handles chargebacks, check out www.paypal.com/chargeback. To learn about PayPal's seller protection program, refer to www.paypal.com/SellerProtection.

Signing Up with PayPal

Once you sign up with PayPal, you can accept payments from customers across the globe. Here's how you go about it:

1. Head to the PayPal website (www.paypal.com), and click the Sign Up link on the home page.
This takes you to PayPal's Sign Up page.
2. Choose your country and language.
3. Choose the type of account you want to create (Personal, Premier, or Business), and then click the Get Started button in the corresponding box.

A *personal account* is ideal if you want to use PayPal to buy items on sites like eBay using your credit card or with funds from your bank account. You can also *accept* money transfers from other PayPal members without having to pay any fees. However, there's a significant catch: Personal accounts can't accept credit card payments. As a result, customers who want to do business with you need to have money in their PayPal accounts (which they get by selling something and receiving a PayPal payment or by transferring money into their account from a linked bank account).

A *premier account* is the best way to run a small business. You can send money (great if you crave a rare movie poster on eBay) and accept any type of payment that PayPal accepts, including both credit and debit cards. You also get to use PayPal's e-commerce tools. However, PayPal charges you a fee for every payment you receive, an amount that varies by sales volume but ranges from 1.9 percent to 2.9 percent of the payment's total value (with a minimum fee of 30 cents). That means that on a \$25 sale, PayPal takes about \$1 off the top. If you accept payments in another currency, you surrender an extra 2.5 percent. To get the full scoop on fees and to see the most current rates, refer to www.paypal.com/fees.

A *business account* is almost identical to a premier account, except that it supports multiple logins. This is the best choice if you have a large business with employees who need to access your PayPal account to help you manage your site.

4. Enter your email address and choose a password, and then fill in your name, address, and phone number.

Make your password complex; you don't want a malicious hacker guessing it and using your PayPal account to go on an electronic buying binge.

Tip: As a general rule, guard your PayPal account information the same way you guard your bank PIN. If you're really paranoid, don't use your PayPal account to buy items on other websites. Don't use your credit card to do so either—electronic eavesdroppers can snag your info and then head out on a first-class cruise to Ibiza.

5. Finally, click "Agree and Create Account" to complete the process.

PayPal sends you an email confirmation immediately. Click the link in the message to activate your account, and then you can start creating PayPal buttons and shopping carts to collect payments (see below for details).

Accepting Payments

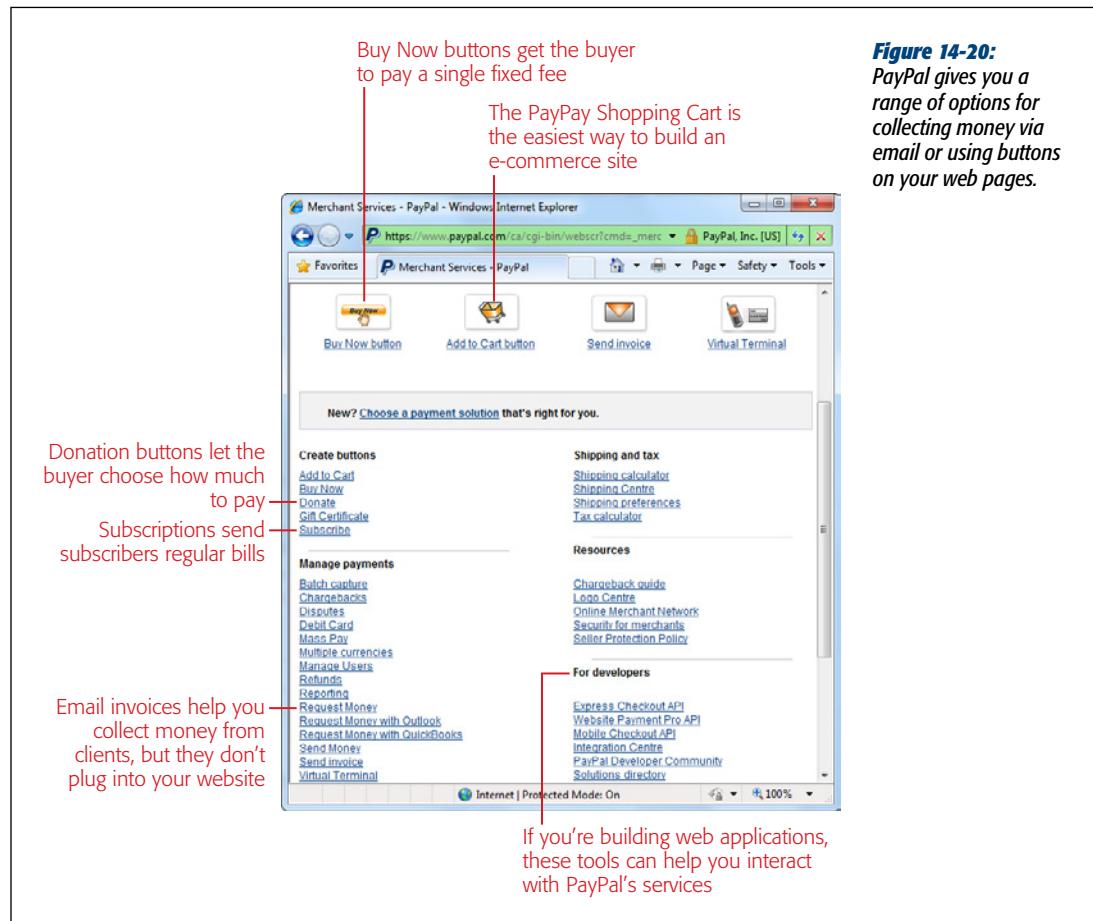
PayPal makes creating e-commerce web pages ridiculously easy. One way is to add a Buy Now button to any page on your site:

1. Go to www.paypal.com, and sign in.
 - After you sign in, you can access several tabs crammed with goodies.
 - Use **My Account** to update your account information, see what recent transactions you made, check your balance, and request a withdrawal.
 - Click **Send Money** to email someone cash (which you supply from your PayPal account, a real-world bank account, or a credit card).
 - Click **Request Money** to send a buyer an email asking for payment.
 - Use the **Create an Invoice** tab to send an email requesting a payment through Pay Pal.

- Click the **Merchant Services** tab to build buttons you can add to your web pages.
- Use **Auction Tools** to specify PayPal as the form of payment for things you sell on eBay. (eBay is still one of the most popular places to set up an e-business.)
- Click the **Products & Services** tab to learn about various PayPal services.

2. Click the **Merchant Services** tab.

Scroll down the page, and you see a variety of tools for collecting money, as explained in Figure 14-20.



3. Click the **Buy Now** link in the **Create Buttons** list.

PayPal displays the page where you configure your Buy Now button's appearance and set the price of your product (Figure 14-21).

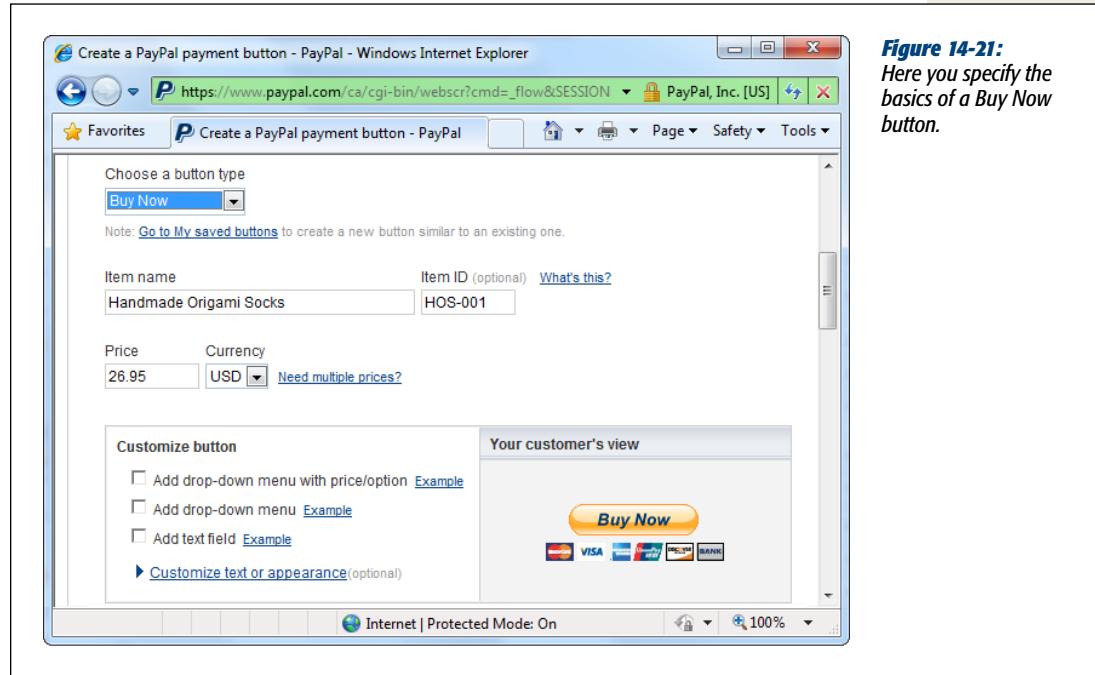


Figure 14-21:
Here you specify the basics of a Buy Now button.

4. Give your item a name and, if you want to keep track of it, a product code. Then supply the price and currency.

Don't worry about locking out international visitors when you set your currency. Credit card companies are happy to charge Canadian customers in U.S. dollars, U.S. customers in euros, and European customers in rupees. Just choose the currency you think your buyers expect to see.

5. If you want to give buyers different options, fill in the information in the "Customize button" box.

You can collect extra buying information from your buyers in three nifty ways:

- Choose **Add drop-down menu with price/option**. This lets you give buyers a list of options for your product, each with a different price (see Figure 14-22). For example, you could let a buyer choose between a plain, premium, or organic tin of hamster food.
- Choose **Add drop-down menu**. This also gives your buyers a list of options, but without changing the product's price. For example, you could use a list like this to let buyers choose the color of the embroidered undergarments they're about to buy.
- Choose **Add text field**. This adds a text box where buyers can type in anything. Use this if you need to collect information that varies, like the name your buyer wants engraved on a magnetic screwdriver.

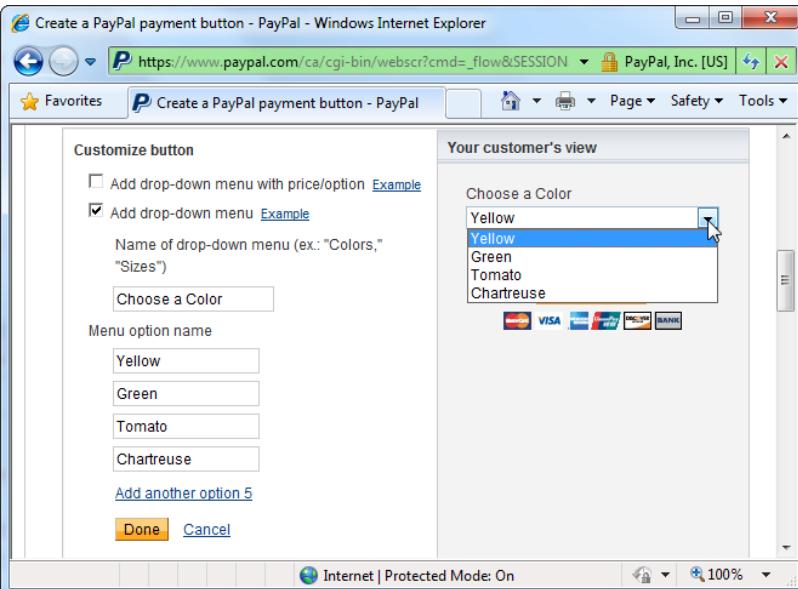


Figure 14-22:
With a drop-down list of options, you can collect additional information about the type of product your visitor wants. This is useful if you offer the same item in multiple sizes or colors (as shown here). To add another color to the list, click the “Add another option” link under the current set of options. After you click Done, PayPal updates the button preview on the right to show you what the list will look like.

6. If you want to change the appearance of a button, click “Customize appearance.”

PayPal gives you limited options for the button’s size and text.

The standard Buy Now button is perfectly usable, but a little plain. If you created a nicer button picture and you’ve uploaded it to your site, supply the URL for the image here. (You can always change the HTML that PayPal generates if you want to use a different button later on.)

7. Scroll down and fill in any additional options you want.

PayPal gives you a heap of extra payment possibilities. You can add a flat fee for shipping and a percentage for sales tax. You can instruct PayPal to track how many items you have in stock and to stop selling your product when it’s sold out—all you do is fill in the number of items you currently have.

And PayPal has an entire section of advanced possibilities, like whether you need a buyer’s address (PayPal assumes you do), if you want your customers to fill in additional comments with their payments (ordinarily, they can’t), and where PayPal should send visitors after they complete or cancel a payment (you can send shoppers to a specific URL on your site, rather than to PayPal’s generic pages).

8. Click Create Button.

PayPal displays see a text box with the markup for your customized Buy Now button. Copy and paste the markup into your web page.

When you create a Buy Now button, PayPal puts everything inside a `<form>` element (explained on page 454). Here's an example of a button for a pair of handmade origami socks based on the last few steps above:

```
<form action="https://www.paypal.com/cgi-bin/webscr" method="post">
  <input type="hidden" name="cmd" value="_s-xclick" />
  <input type="hidden" name="hosted_button_id" value="633788" />
  <table>
    <tr><td>
      <input type="hidden" name="on0" value="Choose a Color" />
      Choose a Color</td></tr><tr><td>
      <select name="os0">
        <option value="Yellow">Yellow</option>
        <option value="Green">Green</option>
        <option value="Tomato">Tomato</option>
        <option value="Chartreuse">Chartreuse</option>
      </select>
    </td></tr>
  </table>
  <input type="image" border="0" name="submit" alt=""
    src="https://www.paypal.com/en_US/i/btn/btn_buynowCC_LG.gif" />
  
</form>
```

If you added any options, you'll see `<select>` and `<option>` elements in the HTML that define the relevant list boxes (page 456). The form also includes the Buy Now button. Clicking it sends the form to PayPal. You can change the button's `src` attribute (bolded in the listing above) to point to a different image file. PayPal inserts the invisible image (`pixel.gif` in the example above) that tracks page views after the code for the Buy Now button. (This tracking technique is the same one Amazon uses.)

Tip: As long as you don't tamper with the `<input>` fields and you keep everything inside the `<form>` elements, you can tweak the markup PayPal creates for you. For example, you can add other elements to the form or gussy it up with a style sheet. Or, you might want to change the layout by removing the invisible table (represented by the `<table>`, `<tr>`, and `<td>` elements) that PayPal uses to organize your button and your options.

So what happens when a shopper clicks the Buy Now button and submits this form? The `action` attribute in the very first line of the code above tells the story: The browser sends the buyer's information to PayPal using the action URL (which is `https://www.paypal.com/cgi-bin/webscr`). As it does, it uses a secure channel to prevent Internet eavesdroppers—that's why the URL starts with "https" instead of "http."

Notice that this form doesn't include key pieces of information, like the product name or price. That's a safety measure designed to prevent troublemakers from tampering with the markup in your web page and paying you less than your products are worth. When PayPal receives the form data, it retrieves the hidden ID value (633788 in the example above), and looks it up in its giant, private database of buttons to identify the relevant product, price, and seller (you).

In fact, the PayPal markup doesn't provide any information about the item you're selling. You put the item name, picture, description, and price into your web page (probably before the Buy Now button). Here's an example:

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

<body>
<h1>Handmade Origami Socks</h1>
<p>
You've waited and they're finally here. Order your own
pair of origami socks for only $26.95 and get them in time
for the holidays. What better way to show your loved ones how
poor your gift giving judgement really is?</p>
<form action="https://www.paypal.com/cgi-bin/webscr" method="post">
...
</form>
</body>
</html>
```

Figure 14-23 shows the result. This example displays the standard PayPal ordering page, but you can customize it with your own logo (see the next section).

Figure 14-23:

Top: The Buy Now button waits patiently on your page.

Bottom: Clicking the Buy Now button starts a secure checkout process using PayPal. Your visitor can pay for an item by credit card, and you both get an email confirming the transaction. Then it's up to you to fulfill your end of the deal.

The figure consists of two screenshots of a web browser. The top screenshot shows a custom product page titled 'Handmade Origami Socks'. It features a small image of origami socks, a descriptive paragraph, and a color selection dropdown set to 'Yellow' with a 'Buy Now' button. The bottom screenshot shows the PayPal 'Choose a way to pay' page for the same item. It displays the order summary: 'Handmade Origami Socks' at '\$26.95', payment method options ('Log in to my PayPal account' or 'Don't have a PayPal account? Pay with your debit or credit card as a PayPal Guest'), and fields for entering credit card information like country, card number, expiration date, and CSC.

Building a Shopping Cart

PayPal's Buy Now button gives you a great way to make a quick sale. But if you dream about an e-commerce empire, you need to create a store where visitors can collect several items at once and pay for them all at the same time.

To give your buyers this kind of convenience, you need a shopping cart, which is a staple of e-commerce websites. The good news is that you don't need to program your own cart—instead, you can use PayPal's prebuilt shopping cart service, which integrates smoothly into your website.

Creating a PayPal shopping cart is remarkably similar to creating a Buy Now button (so if you haven't tried that, you might want to play around with it before you go any further). The basic idea is that you create a separate "Add to Cart" button for each item you sell. You'll get many of the same options you saw when you created a Buy Now button. For example, you can set a price, product code, shipping charges, and so on. The difference is that when visitors click an "Add to Cart" button, PayPal doesn't send them straight to a checkout page. Instead, it displays a shopping cart page in a new window. Visitors can keep shopping until they have everything they want. Then they click a Checkout button to complete their purchase.

To show you how this works, the following example uses the page pictured in Figure 14-24 as a starting point. This example also shows a great use of style-based layout. Check out the downloadable samples—available from the Missing CD page at www.missingmanuals.com/cds/caw3—to try it out for yourself.

The screenshot shows a Windows Internet Explorer window displaying a website for "BrainFood". The title bar reads "Three Panels - Windows Internet Explorer" and the address bar shows "C:\Creating a Website\Chapter 14\BrainFood_Empty.htm". The page content includes a large logo for "BrainFood" and the tagline "Better living through food.". Below the logo are four product descriptions:

- Mystical Brownies**: Transport yourself to another world with these aromatic brownies. May include minced Asian cicadas. (\$27.95)
- Sinful Cookies**: These cookies are officially recognized for their moral-fibre-weakening characteristics. Do NOT bring to church bake sales. (\$17.95)
- Candied Chocolate Eggs**: The only candied eggs to contain 100% real egg. Highly perishable. (\$94.95)
- Double-Take Fruit Flan**: Our award winning fruit flan with high-end plastic fruits. Eat or use to decorate small rooms. (\$32.95)

A sidebar on the right contains a "Contact Us" section with the address "324 Nowhere Place, The Void, Texas 435126 U.S.A.".

Figure 14-24:
Right now, this BrainFood page has a great list of product descriptions, but doesn't give your visitor any way to make an impulse purchase. You can change that by adding a PayPal shopping cart button.

Creating a custom page style

Before you create your shopping cart, there's an extra step you can take to really personalize your payment page. If you're happy with the PayPal standard, feel free to skip straight to the next section. But if you'd like to have your company logo appear on the shopping cart pages, keep reading.

1. If you're not already there, head to www.paypal.com, and sign in.
2. Select the My Account tab, and then click the Profile link just underneath.

The Profile page provides with a slew of information about your preferences, grouped into three main categories: Account Information (who you are and where you live), Financial Information (your bank, credit card, and payment history information), and Selling Preferences (extra options you can use with PayPal's merchant services). In this case, you're interested in the Selling Preferences section.

3. In the Selling Preferences section, click the Custom Payment Pages link.

This takes you to the Custom Payment Page Styles page, where you can set up new page styles or edit existing ones.

4. Click Add to create a new page style.

You start off with only a single page style—the PayPal standard, which sports a basic PayPal logo.

5. Supply information for your page style.

Type a descriptive title into the Page Style Name box to help you remember which style is which.

Use the Header Image URL to point to a picture on your website. This picture is the logo you want to show at the top left of the PayPal shopping cart page. You can use an image up to 750 pixels wide by 90 pixels high.

Note: Because PayPal's shopping cart page is a secure page, when you use a custom logo, the shopper may get a message informing them that there are some insecure items on the page (namely, your picture). If you want to prevent guests from getting this message, talk to your web hosting company about putting your picture on a secure (HTTPS) server.

You can specify color codes for the Header Background Color, Header Border Color, and Background Color of your page. This part is optional; leave it out if you're happy with the standard white.

6. Click Save to store your page style.

Before you commit, you can click Preview to take a sneak peek at what the PayPal payment page looks like.

7. Select your new page style, and then click Make Primary.

Now all your visitors will see your customized checkout page.

Generating the shopping cart buttons

You're ready to build the buttons that add items to an e-shopper's cart. Here's how:

1. If you're not already there, head to www.paypal.com and sign in.
2. Click Merchant Services, and then click the "Add to Cart button" link.

PayPal displays a page where you configure the "Add to Cart" button for a single item.

3. Give your item a name and, if you want to keep track of it, a product code. Then supply the price, currency, and any other relevant information.

These settings are exactly the same as those for a Buy Now button.

4. Click Create Button.

You'll see a text box with the markup for your customized "Add to Cart" button. Copy the text and paste it into your web page. But remember, this "Add to Cart" code applies to a single, specific product. If you have more than one item on a page (as in the BrainFood example), you need to generate multiple buttons. To do so, click "Create similar button" and return to step 3. When you finish generating all the buttons you need and copying each one into your page, continue with the next step.

5. Create a View Cart button.

Your shopping cart solution wouldn't be complete without a button that lets shoppers see what's in their carts (and then head to the virtual checkout counter). To create one, click the "Create a View Cart button" link.

You have virtually no options for the View Cart button, as its purpose is pretty straightforward. You can use the standard View Cart button or supply a URL that points to a button of your own design. Once you make your selection, click Create Button, and then copy the markup into your page along with all the other buttons. Figure 14-25 shows the final result.

Withdrawing Your Money

PayPal safely stashes all your payments in your PayPal account (which is like a virtual bank account). You can see the balance any time. Just log in and click the My Account tab.

If you earn a small amount of money, you may be happy leaving it with PayPal so you can buy other stuff on websites like eBay and www.buy.com. But if you're raking in significant dough, you'll want to transfer some of it to the real world.

The most common approach is to send money to your bank account. To do this, you need to give PayPal your bank account information. PayPal waives its transfer fee as long as your withdrawal meets a certain minimum (like \$150). However, your bank may apply an electronic transaction fee. Depending on the country you live in, PayPal may offer other withdrawal options, too. For example, it may let you transfer money to a debit card or a credit card.

To get started with any of these approaches, log in, click the My Account tab, click the Withdraw link just underneath, and then follow the instructions.

The screenshot shows the BrainFood website homepage. The header features the "BrainFood" logo with the tagline "Better living through food." Below the header, there are four product sections: "Mystical Brownies" (image of a brownie), "Sinful Cookies" (image of cookies), "Candied Chocolate Eggs" (image of colorful eggs), and "Double-Take Fruit Flan" (image of a fruit tart). Each product section includes a brief description and an "Add to Cart" button. To the right of the products is a sidebar with a "Contact Us" section containing address information and a "View Cart" button.

Figure 14-25:

Top: Here's the revised BrainFood page, with shopping cart buttons.

Bottom: After clicking a few "Add to Cart" buttons, here's the shopping cart page your visitors will see (in a separate window). All they need to do is click Check Out to make their purchase.

The screenshot shows the PayPal shopping cart page. The title bar reads "Your shopping cart - PayPal". The page displays a table of items in the cart:

Description	Price	Quantity	Amount
Mystic Brownies Item # Food-001	27.95	1	Update Remove
Candied Chocolate Eggs Item # Food-003	94.95	1	Update Remove

Below the table, it shows an "Item total: \$122.90". At the bottom, there are "Continue Shopping" and "Check Out" buttons.

JavaScript: Adding Interactivity

JavaScript is a simplified programming language designed to beef up web pages with interactive features. It gives you just enough programming muscle to add some fancy effects, but not enough to cause serious damage to your site if your code goes wonky. JavaScript is perfect for creating pop-up windows, embedding animated effects, or modifying the content that appears on your web page. On the other hand, it can't help you build a hot e-commerce storefront; for that, you need the PayPal tools described in Chapter 14 or a server-side programming platform (see page 424).

The goal of this chapter isn't to teach you the details of JavaScript programming—it's to give you enough background so you can find great JavaScript code online, understand it well enough to make basic changes, and then paste it into your pages to get the results you want. Since you can find free JavaScript code on hundreds of websites, these basic skills can come in very handy.

Understanding JavaScript

The JavaScript language has a long history; it first hit the scene with the Netscape Navigator 2 browser in 1995. Internet Explorer jumped on the bandwagon with version 3. Today, all modern browsers run JavaScript, and it's become wildly popular as a result.

Note: JavaScript is thoroughly different from the Java language (although the code sometimes looks similar, because they share some code-writing rules). Java is a full-fledged programming language, every bit as powerful—and complicated—as programming languages like C# and Visual Basic.

Here's what JavaScript can do:

- Dynamically insert new content into a web page or modify an existing HTML element. For example, you can display a personalized message to your visitors (“Hello, Joe!”) or make titles grow and shrink perpetually (see the tutorial on page 444).
- Gather information about the current date, your visitor’s browser, or the content your visitor types into a form. You can display any of this information on a page or use it to make decisions about what your page does next. You could, for example, stop visitors from going any further on your site until they type in an email address.
- React to events that take place in a browser. For example, you can add JavaScript code that runs when a page finishes loading or when a visitor clicks a picture.
- Talk to a program running on a web server. JavaScript can poll the server for a recent stock quote, for example, or for a bunch of records from a company database. This is one task you won’t see covered in this chapter, because it requires some serious programming mojo to write the non-JavaScript code that runs on the web server.

It's just as important to understand what JavaScript *can't* do. JavaScript code is *sandboxed*, which means it's sequestered. A browser locks your JavaScript-containing page into a carefully controlled place in memory known as a sandbox. As a result, it can't perform any potentially risky tasks on your visitor's computer, like sending orders to a printer, accessing files, running other programs, reformatting a hard drive, and so on. This design ensures good security.

Server-Side and Client-Side Programming

To understand how JavaScript fits into the web universe, it's important to understand the two types of programming on the Web.

When you use a search engine like Google or go to an e-commerce site like Amazon, you actually connect to a high-powered piece of software known as a *server-side application*, which runs on a web server. When you visit one of these sites, you send the server-side program information, like the keywords you want to search for or the book you want to buy. The program, in turn, consults a massive database and spits out some HTML that creates the page you see in your browser.

Server-side applications rule the web world, because there's virtually nothing they can't do. However, they're difficult to program. Not only do developers need to worry about getting the program to generate HTML for a browser, they also need to make sure the program can run all kinds of complex code and tap giant databases—and they need to do it so that the site performs just as well when millions of people view it as it does when only one person visits it. This is hard work, and it's best handled by the poor souls we call programmers.

Client-side applications, on the other hand, use a completely different model. They embed small, lightweight programs inside an ordinary HTML page. When a browser

downloads the page, the browser itself runs the program (assuming your security settings or compatibility issues haven't disabled the program). Client-side programs are much less powerful than those on the server side—they can't reliably poll the huge databases stored on web servers, for example, and for security reasons they can't directly change most things on your computer. However, they're much simpler to create.

Note: The distinction between server-side and client-side programs is sometimes muddled by the fact that top-flight websites use both. For example, Amazon uses fine-tuned JavaScript for its pop-up menus and "Look Inside" feature, which lets you browse a book. However, all the serious stuff, like tracking orders, processing credit cards, and storing customer reviews, happens on the web server.

Scripting Languages

The world of client-side programming essentially falls into two categories. On one side, you find *rich programming environments*. These technologies require a browser plug-in and provide advanced features that are similar to those in desktop applications. The most popular example is Adobe Flash, but Microsoft Silverlight is also a worthy competitor. If you've ever watched a video or played a free online game in your browser (see Figure 15-1), you've almost certainly used one of the two. And if you've ever cursed out loud at an animated ad that suddenly takes over your entire browser window, you have one of them to thank for that as well.

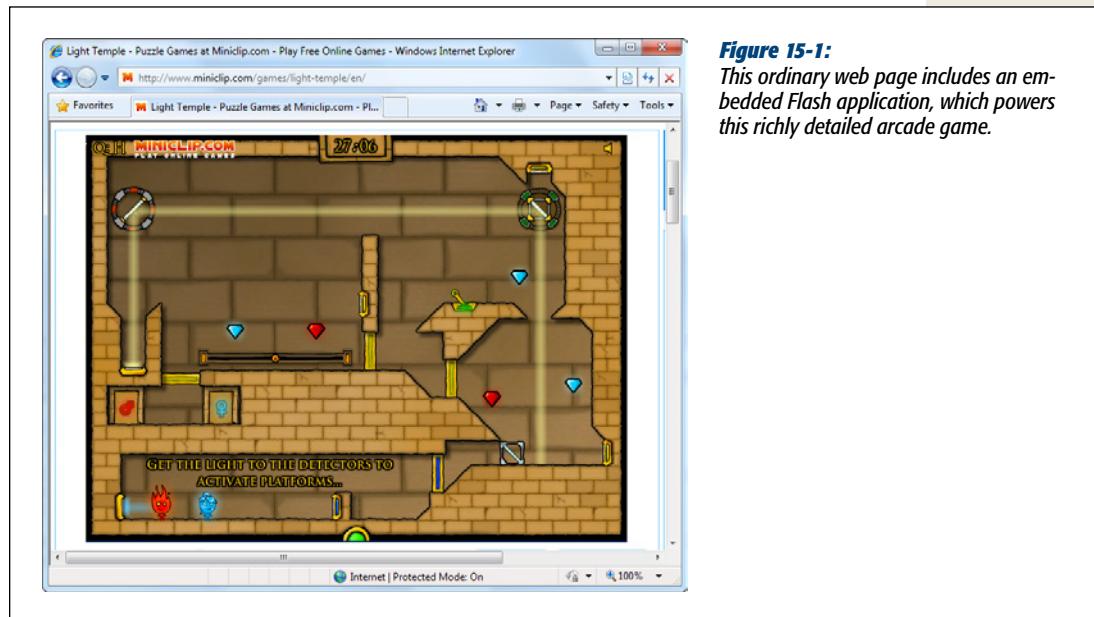


Figure 15-1:

This ordinary web page includes an embedded Flash application, which powers this richly detailed arcade game.

On the other side are scripting languages like JavaScript. JavaScript isn't as advanced as Flash and Silverlight, but its simplicity works to its advantage in three ways:

- You don't need to write an entire program to use a scripting language. Instead, you create a *script*—a short scrap of text you can place directly in your web page. This script gives a browser a series of instructions, like “scroll that heading from left to right” or “pop up an ad for fudge-flavored toothpicks in a new window.”
- Even if you don't know all the ins and outs of the JavaScript language, you can often copy and paste a cool script from a free website to get instant gratification.
- Because JavaScript doesn't need a plug-in, and because it's guaranteed to work on every browser, big businesses can rely on it to add a bit of pizzazz to their pages, without leaving anyone out (Apple's iPhone and iPad don't run plug-ins like Flash and Silverlight, for example).

Today, JavaScript is the only scripting language with any true measure of browser support, so it's the only scripting language anyone uses on the Web.

HTML5 PREVIEW

Not Your Father's JavaScript

JavaScript started out life as a gimmicky toy. Even when code-powered websites like Amazon and eBay first took off, JavaScript played a minor role.

Today, however, e-commerce sites use JavaScript all the time. When big companies create web applications, they use one of a variety of programming platforms on their server (including ASP.NET, PHP, JSP, Perl, Ruby, and more), but they stick with JavaScript to power the client-side parts. And super-genius developers are constantly expanding the boundaries of what web pages can do by combining JavaScript and server-side code. Google now uses that blend—not Flash—for advanced web applications like Google Maps and Gmail.

This trend is accelerating with HTML5. The new version of the HTML language comes bundled with a pile of next-generation standards that extend JavaScript. In fact, that's a large part of the reason why HTML5 attracted so much attention and went from outsider status to an accepted standard in record time (see the story on page 25). When HTML5 is finalized and implemented in modern browsers, developers can use JavaScript to locate a web surfer's geographic location, draw dynamic graphics, and even take control of webcams and microphones. So consider the time you spend learning JavaScript today an investment in the future of web development.

JavaScript 101

Now that you've learned a bit about JavaScript and why it exists, it's time to dive in and start creating your first real script.

The `<script>` Element

Every script starts with a `<script>` block you slot somewhere into an HTML document. Really, you have only two options:

- The `<body>` section. Put scripts you want your browser to run right away in the `<body>` section of your HTML. The browser runs your script as soon as it reaches the `<script>` element. If you put your script at the beginning of the `<body>` section, your browser processes the script before it displays the page.

Tip: Usually, JavaScript fans put their scripts at the *end* of the `<body>` section. That way, you avoid errors that might occur if you use a script that relies on another part of the page, and the browser hasn't read that section yet. Because browsers read an entire page quickly, these scripts execute almost immediately.

- The `<head>` section. If you place an ordinary script in the `<head>` section of your HTML document, it runs immediately, before the browser processes any part of the markup. However, it's more common to use the `<head>` section for scripts that contain *functions* (see page 434). Functions don't run immediately—instead, you summon them when your visitor takes some kind of action on a page, like moving a mouse.

Note: You can place as many `<script>` blocks in a web page as you want.

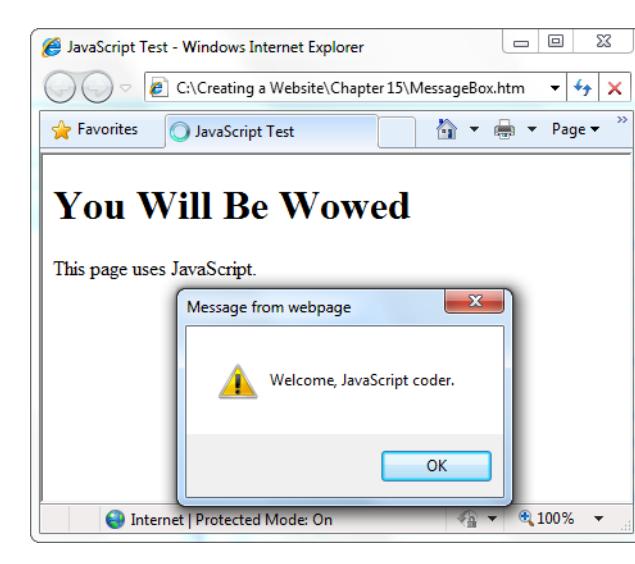
A typical script block consists of a series of programming instructions. To get a handle on how these instructions work, consider the following example, which displays a JavaScript alert box on your page:

```
<!DOCTYPE html>

<html>
  <head>
    <title>JavaScript Test</title>
  </head>

  <body>
    <h1>You Will Be Wowed</h1>
    <p>This page uses JavaScript.</p>
    <script type="text/javascript">
      alert("Welcome, JavaScript coder.")
    </script>
  </body>
</html>
```

This script pops up a window with a message, as shown in Figure 15-2. When you click OK, the message disappears, and it's back to life as usual for your web page.

**Figure 15-2:**

Because you positioned the `<script>` element for this page at the end of the HTML markup, the browser displays all the HTML first and then pops up the alert box. If you put the `<script>` element at the beginning of the `<body>` section (or in the `<head>` section), the alert box would appear earlier, while the page is still blank. The browser would then wait until you clicked OK before reading the rest of the HTML page and displaying its contents.

Like all scripts, the script in this example is wrapped inside a `<script>` element:

```
<script>
  ...
</script>
```

You're probably wondering exactly how this script works its magic. When a browser processes the script, it runs all the code, going one line at a time. In this case, there's only one line:

```
alert("Welcome, JavaScript coder.")
```

This line uses a built-in JavaScript function called `alert`. A *function* is a programming routine consisting of one or more lines of code that performs a certain task. JavaScript has many built-in functions, but you can also build your own.

JavaScript's `alert()` function requires one piece of information, known as an *argument* in programmer-speak. In this case, that piece of information is the text you want the alert box to display. If you want to see an ordinary number, say 6, you could type it in as is—that is, you don't need to put it in quotes. But with text, there's no way for a browser to tell where text starts and stops. To compensate for this in JavaScript, you put text inside single apostrophe quotes ('') or double quotation marks (""), as in the example above.

Note: In programmer-speak, a distinct piece of text used in a program is called a *string*. "The friendly fox," "a," and "Rumpelstiltskin" all qualify as strings.

That's it. All this simple script does is call JavaScript's `alert()` function. (Spend enough time around programmers and JavaScript fans, and you'll soon learn that "call" is the preferred way to describe the action that triggers a function.) The `alert()` function does the rest, popping up the correct-sized window and waiting for your visitor to click OK.

Note: To write this script, you need to know that there's an `alert()` function ready for you to use—a fact you can find out on one of the many JavaScript tutorial sites.

Based on what you now know, you should be able to change this script to:

- Display a different message (by changing the argument)
- Display more than one message box, one after the other (by adding more lines in your `<script>` block)
- Display the message box before your browser displays the web page (by changing the position of the `<script>` block)

It's not much to keep you occupied, but the `alert()` function does show you how easily you can get started using and changing a simple script.

GEM IN THE ROUGH

Dealing with Internet Explorer's Paranoia

If you run the alert example above in the Firefox browser, you'll find that everything works seamlessly. If you run it in Internet Explorer, you won't get the same satisfaction. Instead, you'll see a security warning in a yellow bar at the top of the page. Until you click that bar and then choose Allow Blocked Content, your JavaScript code won't run.

At first glance, IE's security warning seems like a surefire way to scare off the bravest web visitor. But you don't need to worry; the message is just part of the quirky way Internet Explorer deals with web pages that you store on your hard drive. When you access the same page over the Web, Internet Explorer won't raise the slightest objection.

That said, the security warning is still an annoyance while you're testing your web page, because it forces you to keep

explicitly telling the browser to allow the page to run JavaScript. To avoid the security notice altogether, you can tell Internet Explorer to pretend you downloaded your page from a web server. You do this by adding a special comment called the *Mark of the Web*. You place this comment immediately after the `<html>` element that begins your page:

```
<html>
<!-- saved from url=(0014)about:internet -->
```

When IE sees the Mark of the Web, it treats the page as though it came from a web server, skipping the security warning and running your JavaScript code without hesitation. To all other browsers, the Mark of the Web just looks like an ordinary HTML comment.

Scripts and HTML

As you've seen, HTML uses the `<script>` block to hold JavaScript code. But understand that `<script>` is just one more HTML element, and like all HTML elements, it needs to follow the language's rules. One of those rules limits what you can put

inside the `<script>` block. HTML forbids certain characters, like the infamous angle brackets, because they have a special meaning in the HTML language.

Unfortunately, special characters *do* sometimes crop up in code. You could replace them with character entities (page 130), but that makes it far more difficult for other people to read, understand, and edit your program. A better solution is to wrap the entire script in something called a *CDATA section*—a specialized region of your HTML document that suspends the usual HTML rules. Here's how:

```
<script>
//<![CDATA[
    alert("Welcome, JavaScript coder.")
//]]>
</script>
```

This technique is a bit ugly, but it does the trick. It's best not to think about it too deeply and just get into the habit of wrapping any lengthy block of script code in a CDATA section. The one exception is JavaScript code you put in a separate file (page 439)—your browser won't interpret it as HTML, so you don't need to wrap it in a CDATA section.

Note: For simplicity, this chapter uses a CDATA section only when absolutely necessary—in other words, when the script contains special characters that HTML ordinarily wouldn't allow.

UP TO SPEED

Spaces and Line Breaks in JavaScript

JavaScript code is quite tolerant of extra spaces. In this chapter, most of the examples use some sort of indenting to help you see the structure of the code. But as with HTML, you don't absolutely have to add these spaces.

The only rule in JavaScript is that every code statement needs to be on a separate line. You can get around this limitation by using the line-termination character, which is a semicolon (;). For example, here's how you can compress three code statements onto one line:

```
alert("Hi"); alert("There"); alert("Dude");
```

Each semicolon designates the end of a code statement. (This strange convention comes from the Bizarro world of C and Java.)

If you don't want to put more than one code statement on the same line, you don't need the semicolons. However, you're free to add them. In fact, if you download a script from the Web, you'll usually find these optional awkward widowed syllable.

Browsers that don't support JavaScript

It's rare, but some browsers will recognize the `<script>` element but refuse to execute your code. This can happen if a browser doesn't support JavaScript (for example, a dusty text-only browser like Lynx) or if JavaScript has been switched off (which is possible in paranoid corporate environments, but still very rare).

To deal with the occasional situation like this, you can use the `<noscript>` element, which lets you supply alternate HTML content. You place the `<noscript>` element immediately after the `<script>` element. Here's an example that displays a paragraph of text for browsers that lack JavaScript support:

```
<script>
  alert("Welcome, JavaScript coder.")
</script>
<noscript>
  <p>Welcome, non-JavaScript-enabled browser.</p>
</noscript>
```

Variables

Every programming language includes the concept of *variables*, which are temporary containers that store important information. Variables can store numbers, objects, or pieces of text. As you'll see throughout this chapter, variables play a key role in many scripts, and they're a powerful tool in any programmer's arsenal.

Declaring variables

To create a variable in JavaScript, you use the `var` keyword, followed by the name of the variable. You can choose any name that makes sense to you, as long as you're consistent (and avoid spaces or special characters). This example creates a variable named `myMessage`:

```
var myMessage
```

To store information in a variable, you use the equal sign (`=`), which copies the data on the right side of the equal sign into the variable on the left. Here's an example that puts some text into `myMessage`:

```
myMessage = "Everybody loves variables"
```

Remember, you need to use quotation marks whenever you include a text string. In contrast, if you want to copy a *number* into a variable, you don't need quotation marks:

```
myNumber = 27.3
```

Note: JavaScript variables are case-sensitive, which means a variable named `myMessage` differs from one named `MyMessage`. If you try to use them interchangeably, you'll wind up with an error message (if your browser is nice) or a bizarre mistake in the page (which is usually what happens).

You'll often want to create a variable and fill it with useful content, all in the same step. JavaScript lets you do so if you put an equal sign immediately after the variable name when you declare it:

```
var myMessage = "Everybody loves variables"
```

To make matters a little confusing, JavaScript lets you refer to variables you haven't yet declared. Doing so is considered extremely bad form and is likely to cause all

sorts of problems. However, it's worth knowing that these undeclared variables are permissible, because they're the source of many an unexpected error.

Modifying variables

One of the most useful things you can do with numeric variables is perform *operations* on them to change your data. For example, you can use arithmetic operators to perform mathematical calculations:

```
var myNumber = (10 + 5) * 2 / 5
```

These calculations follow the standard order of operations (parentheses first, then multiplication and division, then addition and subtraction). The result of this calculation is 6.

You can also use operations to join together multiple pieces of text into one long string. In this case, you use the plus (+) operator:

```
var firstName = "Sarah"  
var lastName = "Smithers"  
var fullName = firstName + " " + lastName
```

Now the *fullName* variable holds the text “Sarah Smithers.” (The “ ” in the code above tells JavaScript to leave a space between the two names).

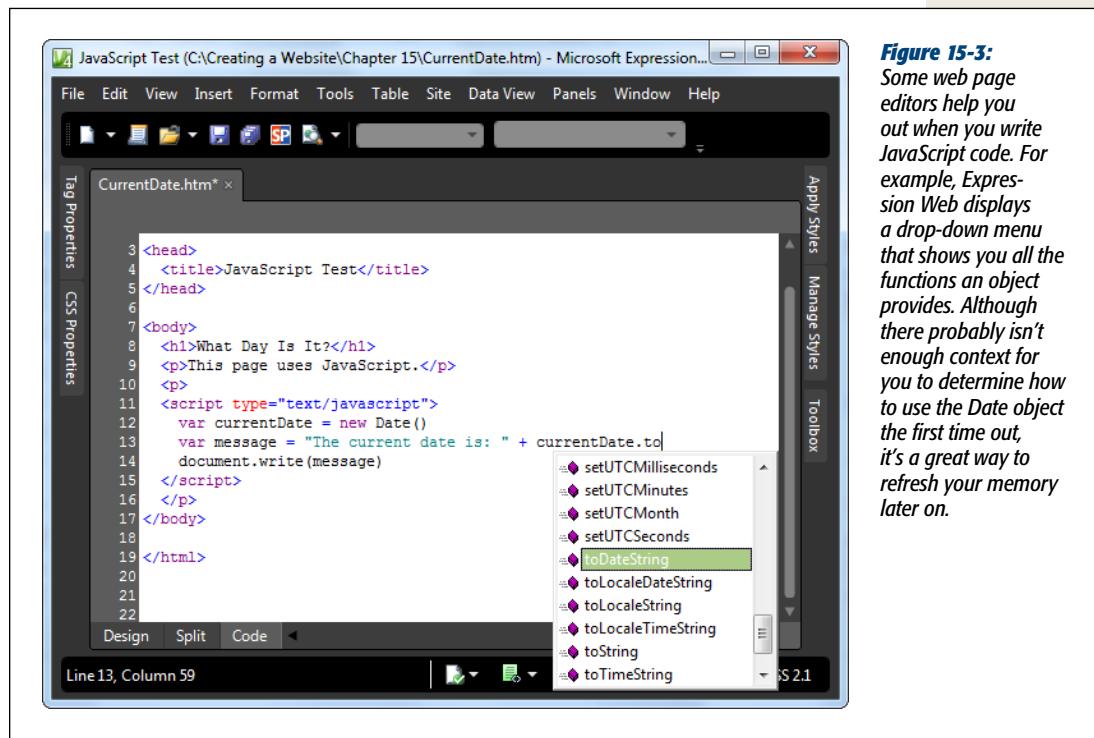
An example with variables

Although you'd need to read a thick volume to learn everything there is to know about variables, you can pick up a lot from a simple example. The following script inserts the current date into a web page. The example numbers each line of code to make it easy to reference.

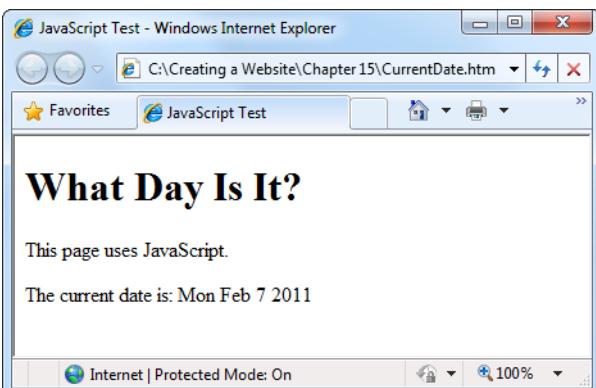
```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  <title>JavaScript Test</title>  
</head>  
  
<body>  
  <h1>What Day Is It?</h1>  
  <p>This page uses JavaScript.</p>  
  <p>  
    <script>  
      1      var currentDate = new Date()  
      2      var message = "The current date is: "  
      3      message = message + currentDate.toDateString()  
      4      document.write(message)  
    </script>  
    </p>  
  </body>  
</html>
```

Here's what's happening, line by line:

1. This line creates a new variable named *currentDate*. It fills the *currentDate* variable with a new Date object (see number 3 below). You'll know JavaScript is creating an object when you see the keyword *new*. (You'll learn more about objects on page 440; for now, it's enough to know that objects come with built-in functions that work more or less the same way as the functions you learned about earlier.)
2. This line creates a new variable named *message* and fills it with the beginning of a sentence that announces the date.
3. This line adds some new text to the end of the message you created in line 2. This text comes from the *currentDate* object. The tricky part is understanding that the *currentDate* object comes with a built-in function, *toString()*, that converts the date information it gets from your computer into a piece of text suitable for display in a browser (see Figure 15-3). Once again, this is the kind of detail you can only pick up by studying a good JavaScript reference.



4. This line uses JavaScript's *document* object, which has a function named *write()*. The *write()* function displays a piece of text on a web page at the current location. The final result is a page that shows your welcome message (see Figure 15-4).

**Figure 15-4:**

The `document.write()` command inserts text directly into a page, wherever you position the script block. In this case, the command displays the current date.

Scripts can get much more complex than this. For example, they can use loops to repeat a single action several times, or scripts can make decisions using conditional logic. You'll see examples of some of these techniques later in this chapter, but you won't get a blow-by-blow exploration of the JavaScript language—in fact, that would require a small book of its own. If you want to learn more, check out a book like *JavaScript: The Missing Manual*.

Functions

So far, you've seen simple scripts that use only a few lines of code. More realistic JavaScript scripts can run to dozens of lines and if you're not careful, they can grow into a grotesque tangle that leaves the rest of your page difficult to edit. To control the chaos, smart JavaScripters almost always use *custom functions*.

A *function* is a series of code instructions you group together and give a name. In a way, functions are like miniature programs, because they can perform a series of operations. The neat thing about functions is that you need to create them only once, and you can reuse them over and over again.

Declaring a function

To create a JavaScript function, start by deciding what your function is going to do (like display an alert message), and then choose a suitable name for it (like `ShowAlertBox`). As with most things in the programming world, function names can't have any spaces or special characters.

Armed with this information, you're ready to put a `<script>` block in the `<head>` section of your page. But this `<script>` block looks a little different from the examples you've seen so far. Here's a complete function that shows an alert box with a predefined message:

```
<script>
  function ShowAlertBox() {
    alert("I'm a function.")
  }
</script>
```

To understand what's going on here, it helps to break down this example and consider it piece by piece.

Every time you declare a function, you start with the word *function*, which tells JavaScript what you're up to.

function

Then, you name your function and add two parentheses. You use the parentheses to send extra information to your function, as you'll see shortly.

function ShowAlertBox()

At this point, you've finished *declaring* the function. All that remains is to add the code to the function that actually makes it work. To do this, you need the funny curly braces shown in the alert box function above. The { brace indicates the start of your function code and the } brace indicates the end of it. You can put as many lines of code as you want in between.

One tricky part of function-writing is the fact that JavaScript sets notoriously loose standards for line breaks. That means you can create an equivalent JavaScript function and put the curly braces on their own line, like this:

```
<script>
  function ShowAlertBox()
  {
    alert("I'm a function.")
  }
</script>
```

But don't worry—both functions work exactly the same way.

Tip: You can put as many functions as you want in a single `<script>` block. Just add them one after the other.

Calling a function

Creating a function is only half the battle. On their own, functions don't do anything. You have to *call* the function somewhere in your page to actually run the code. To call a function, you use the function name, followed by parentheses:

`ShowAlertBox()`

Note: Don't leave out the parentheses after the function name. Otherwise, browsers will assume you're trying to use a variable rather than call a function.

You can call ShowAlertBox() anywhere you'd write ordinary JavaScript code. For example, here's a script that displays the alert message three times in a row to really hassle your visitors:

```
<script>
  ShowAlert()
  ShowAlert()
  ShowAlert()
</script>
```

This is the same technique you used to call the alert() function earlier. The difference is that alert() is built into JavaScript, while ShowAlertBox() is something you created yourself. Also, the alert() function requires one argument, while ShowAlertBox() doesn't use any.

Functions that receive information

The ShowAlertBox() function is beautifully simple. You simply call it, and it displays an alert box with the message you supplied. Most functions don't work this easily. That's because in many cases you need to send specific information to a function, or take the results of a function and use them in another operation.

For example, imagine you want to display a welcome message with some standard information in it, like the current date. But you also want the flexibility to change part of the message by substituting your own witty words each time you call the function. To do so, you need a way to call a function *and* to supply a text string with your message in it.

To solve this problem, you can create a ShowAlertBox() function that accepts a single *parameter*. This parameter represents the customized text you want to incorporate into your greeting. To add the parameter, you must first give it a name, say *customMessage*, and put it in parentheses after the function name, like so:

```
function ShowAlertBox(customMessage) {
  ...
}
```

Note: Technically, the pieces of information that a function receives (in this case, that's *customMessage*) are called *parameters*. When you call a function that has parameters, you pass the function one value for each parameter. The value you supply is called an *argument*.

In other words, the same piece of information is called a parameter from the function's point of view, and an argument from the calling code's point of view. Sometimes, you'll hear the terms "parameter" and "argument" used interchangeably, but now you know the official difference.

There's no limit to how many pieces of information a function can accept. You just need to separate each parameter with a comma. Here's an example of the ShowAlertBox() function with three parameters, named *messageLine1*, *messageLine2*, and *messageLine3*:

```
function ShowAlertBox(messageLine1, messageLine2, messageLine3) {
    ...
}
```

Here's another example that shows a finished ShowAlertBox() function. It accepts a single parameter named customMessage, and it uses the customMessage parameter to create the text it displays in the alert box:

```
<script>
1   function ShowAlertBox(customMessage)
2   {
3       // Get the date.
4       var currentDate = new Date()
5
6       // Build the full message.
7       var fullMessage = "*** IMPORTANT BULLETIN ***\n\n"
8       fullMessage += customMessage + "\n\n"
9       fullMessage += "Generated at: " + currentDate.toTimeString() + "\n"
10      fullMessage += "This message courtesy of MagicMedia Inc."
11
12      // Show the message.
13      alert(fullMessage)
14  }
</script>
```

Here are some notes to help you wade through the code:

- Any line that starts with // is a comment (see lines 3 and 6). Good programmers include lots of comments to help others understand how a function works (and help themselves remember what they did during a late-night coding binge). The browser ignores them.
- To put line breaks into an alert box, use the code \n (lines 7, 8, and 9). Each \n is equivalent to one line break. (This rule is for message boxes only. When you want a line break in HTML, use the familiar
 element.)
- To build the text for the *fullMessage* variable (lines 7 to 10), the code uses a shortcut in the form of the += operator. This operator automatically takes whatever's on the *right* side of the equal sign and pastes it onto the end of the variable on the *left* side. In other words, this:

```
8       fullMessage += customMessage + "\n\n"
```

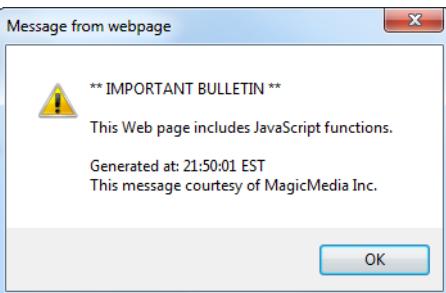
is equivalent to this longer line:

```
8       fullMessage = fullMessage + customMessage + "\n\n"
```

Using this function is easy. Just remember that when you call it, you need to supply one argument for each parameter, separating them with a comma. In the case of the ShowAlertBox() function above, you only need to supply a single value for the customMessage variable. Here's an example:

```
<script>
  ShowAlertBox("This web page includes JavaScript functions.")
</script>
```

Figure 15-5 shows the result of this script.

**Figure 15-5:**

This message is built out of several pieces of text, one of which you supplied as an argument to the `ShowAlertBox()` function.

Functions that return information

Parameters let you send information *to* a function. You can also create functions that send information *back* to the script code that called them. The key to doing this is the `return` command, which you put right at the end of your function. The `return` command ends the function immediately, and spits out whatever information your function generates.

Of course, a sophisticated function can accept *and* return information. For example, here's a function that multiplies two numbers (the `numberA` and `numberB` parameters) and returns the result to anyone who's interested:

```
<script>
    function MultiplyNumbers(numberA, numberB)
    {
        return numberA * numberB
    }
</script>
```

Here's how you use this function elsewhere on your web page:

```
<p>The product of 3202 and 23405 is
<script>
    var product = MultiplyNumbers(3202, 23405)
    document.write(product)
</script>
</p>
```

This HTML includes a single line of text, followed by a block of script code. The script code calls the `MultiplyNumbers()` function, gets the result (the number 74942810), and stuffs it in a variable named `product` for later use. The code then uses the `document.write()` command to display the contents of the `product` variable on the page. The final result is a paragraph with this text:

The product of 3202 and 23405 is 74942810

To use a typical script you get from the Web, you need to copy one or more functions into your page. These functions are likely to look a lot more complex than what you've seen so far. However, now that you understand the basic structure of a function,

you can wade through the code to get a fundamental understanding of what's taking place (or to at least pinpoint where the action goes down).

External Script Files

Reusing scripts inside a web page is neat, but did you know that you can share scripts *between* individual pages and even among different websites? You simply put your script into an external file and then link to it. This procedure is similar to the way you learned to link external style sheets back in Chapter 6.

For example, imagine you perfect the ShowAlertBox() routine so that it performs a complex task exactly the way you want it to, but it requires a couple of dozen lines of code. To simplify your life and your HTML document, you create a new file to store that script.

Script files are always plain text files. Usually, they have the extension `.js` (for JavaScript). You put all your code inside a script file, but you don't include the `<script>` element. For example, you could create this JavaScript file named `ShowAlert.js`:

```
function ShowAlertBox()
{
    alert("This function is in an external file.")
}
```

Now save the file, and put it in the same folder as your web page. In your web page, define a script block, but don't supply any code. Instead, add the `src` attribute and indicate the script file you want to link to:

```
<script src="ShowAlert.js">
</script>
```

When a browser comes across this script block, it requests the `ShowAlert.js` file and treats it as though the code were right inside the page. Here's a complete HTML test page that uses the `ShowAlert.js` file. The script in the body of the page calls the `ShowAlertBox()` function:

```
<!DOCTYPE html>

<html>

<head>
    <title>Show Alert</title>
    <!-- Make all the functions in the ShowAlert.js file
        available in this page. Notice there's no actual content here. -->
    <script src="ShowAlert.js">
    </script>
</head>

<body>
    <!-- Test out one of the functions. -->
    <script>
        ShowAlertBox()
    </script>
</body>

</html>
```

There's no difference in the way an embedded or external script works. However, storing your scripts in separate files helps keep your website organized and makes it easy to reuse scripts across several pages. In fact, you can even link to JavaScript functions on another website—just remember that the `src` attribute in the `<script>` block needs to point to a full URL (like `http://SuperScriptSite.com>ShowAlert.js`) instead of just a file name. Of course, this technique is risky because the website owner might rename, move, or modify the JavaScript file. If you really want to use the code, it's far better to copy it to your own server to avoid this problem.

Note: Using separate script files doesn't improve your security one iota. Because anyone can request your script file, a savvy web visitor can figure out what scripts your page uses and take a look at them. So never include any code or secret details in a script that you don't want the world to know about.

Dynamic HTML

JavaScript underwent a minor revolution in the late 1990s, adding support for a set of features called *Dynamic HTML* (also shortened to DHTML). Dynamic HTML isn't a new technology—it's a fusion of three distinct ingredients:

- Scripting languages like JavaScript, which let you write code
- The CSS (Cascading Style Sheet) standard, which lets you control the position and appearance of an HTML element
- The HTML *document object model* (or DOM), which lets you treat an HTML page as a collection of *objects*

The last point is the most important. Dynamic HTML exposes the page as a collection of *objects*. This is a radical shift in web programming. Dynamic HTML treats each HTML element, including images, links, and even the lowly paragraph, as a separate programming ingredient that your JavaScript code can play with. Using these objects, you can change what each element looks like or even where your browser places them on a page.

HTML Objects

Clearly, Dynamic HTML requires a whole new way of thinking about web page design. Your scripts no longer look at your web page as a static block of HTML. Instead, they see a combination of *objects*.

UP TO SPEED

Understanding Objects

In many programming languages, including JavaScript, everything revolves around objects. So what, exactly, is an object?

In the programming world, an object is nothing more than a convenient way to group some related features or information. For example, say you want to change the picture shown in an `` element on a web page (which is useful if you want to write a script that flashes a series of images). The easiest way to interact with an `` element in JavaScript is to use the corresponding *image* object. In effect, the image object is a container holding all sorts of potentially useful information about what's happening inside an `` element (including its dimensions, its position, the name of the image file associated with it, and so on). The image object also gives you a way to manipulate the `` element—that is, to change some or all of these details.

For example, you can use an image object to get information about the image, like this:

```
document.write("The tooltip says" + image.title)
```

You can even change one of these details. For example, you can modify the actual image that an `` element shows by using this code:

```
image.src = "newpic.jpg"
```

You'll know an object's at work by the presence of a dot (.) in your code line. The dot separates the name of the variable (the first part) from one of the built-in functions it provides (called *methods*), or from one of the related variables (called *properties*). You always put methods and properties after a period.

In the previous examples, *src* and *title* are two of the image object's properties. In other words, the code `image.src = "newpic.jpg"` is equivalent to saying “Hey, Mr. Object named Image: I have a new picture for you. Change your *src* to point to *newpic.jpg*.“

Programmers embraced objects long ago because they're a great way to organize code conceptually (not to mention a great way to share and reuse it). You might not realize it at first, but working with the image object is actually easier than memorizing a few dozen different commands that manipulate the image itself.

Before you can manipulate an object on your web page, you need a way to uniquely identify it. Once you do that, your code can find the object it needs. The best choice for identifying objects is the *id* attribute. Add this attribute to the start tag for the element you want to manipulate and choose a unique name, as shown here:

```
<h1 id="PageTitle">Welcome to My Page</h1>
```

Once you give your element a unique ID, you can easily locate that object in your code and have JavaScript act on it.

JavaScript includes a handy trick for locating an object: the `document.getElementById()` method. Basically, `document` is an object that represents your whole HTML document. It's always available and you can use it any time you want. This document object, like any object worthy of its name, gives you some handy properties and methods. The `getElementById()` method is one of the coolest: It scans a page looking for a specific HTML element.

Note: In the example on page 432, you saw the document object at work on a different task—displaying information on a web page. To accomplish this feat, the script used the write() method of the document object.

When you call the document.getElementById() method, you supply the ID of the HTML element you’re looking for. Here’s an example that digs up the object for an HTML element with the ID *PageTitle*:

```
var titleObject = document.getElementById("PageTitle")
```

This code gets the object for the <h1> element shown earlier and stores it in a variable named *titleObject*. By storing the object in a variable, you can perform a series of operations on it without having to look it up more than once.

So what, exactly, can you do with HTML objects? To a certain extent, the answer depends on the type of element you’re working with. For example, if you have a hyperlink, you can change its URL. If you have an image, you can change its source. And there are some actions you can take with almost all HTML elements, like changing their style or modifying the text that appears between the beginning and ending tags. As you’ll see, you’ll find these tricks useful in making your pages more dynamic—for example, you can change a page when a visitor takes an action, like clicking a link. Interactions like these make visitors feel as though they’re using an intelligent, responsive program instead of a plain, inert web page.

Here’s how you modify the text inside the just-mentioned <h1> element, for example:

```
titleObject.innerHTML = "This Page Is Dynamic"
```

If you use this code in a script, the header’s text changes as soon as your browser runs the script.

This script works because it uses the *property* named *innerHTML*, which sets the content that’s nested inside an element (in this case, the <title> element). Like all properties, *innerHTML* is just one aspect of an HTML object you can alter. To write code statements like this, you need to know what properties JavaScript lets you play with. Obviously, some properties apply to specific HTML elements only, like the *src* attribute of an image. But modern browsers boast a huge catalog of DOM properties you can use with just about any HTML element. Table 15-1 lists some of the most useful.

Tip: To get the properties that a specific HTML element supports, check out the reference at www.w3schools.com/html/dom_reference.asp.

Currently, the example above works in two steps (getting the object, and then manipulating it). Although this two-step maneuver is probably the clearest approach, it's possible to combine these two steps into one line, which scripts often do. Here's an example:

```
document.getElementById("PageTitle").innerHTML = "This Page Is Dynamic"
```

Remember, the advantage to getting an object first is that you can change several properties one after the other, without needing to look up the HTML object using `getElementById()` each time.

Table 15-1. Common HTML object properties.

Property	Description
<code>className</code>	Lets you retrieve or set the <code>class</code> attribute (see page 147). In other words, this property determines what style (if any) this element uses. Of course, you need to define this style in an embedded or linked style sheet, or you'll end up with the plain-Jane default formatting.
<code>innerHTML</code>	Lets you read or change the HTML inside an element. This property is insanely useful, but it has two quirks. First, you can use it on all HTML content, including text and tags. So if you want to put bold text inside a paragraph, you can set <code>innerHTML</code> to <code>Hi</code> . Special characters aren't welcome—you need to replace them with the character entities described on page 130. Second, when you set <code>innerHTML</code> , you replace all the content inside this element, including any other HTML elements. So if you set the <code>innerHTML</code> of a <code><div></code> element that contains several paragraphs and images, all these items disappear, to be replaced by your new content. If you want to modify a specific piece of a paragraph, wrap that piece in a <code></code> element.
<code>parentElement</code>	Provides the HTML object for the element that contains this element. For example, if the current element is a <code></code> element in a paragraph, this gets the object for the <code><p></code> element. Once you have this object, you can modify the paragraph. Using this technique (and other similar techniques in Dynamic HTML), you can jump from one element to another.
<code>style</code>	Bundles together all the CSS attributes that determine the appearance of the HTML element. Technically, the <code>style</code> property returns a full-fledged style object, and you need to add another dot (.) and the name of the style attribute you want to change, as in <code>myObject.style.fontSize</code> . You can use the <code>style</code> object to dictate colors, borders, fonts, and even positioning.
<code>tagName</code>	Provides the name of the HTML element for this object, without the angle brackets. For example, if the current object represents an <code></code> element, this returns the text <code>img</code> .

Using HTML objects in a script

The easiest way to come to grips with how HTML objects work is to look at an example. The web pages shown in Figure 15-6 include a paragraph that continuously grows and then shrinks, as your code periodically tweaks the font size.

**Figure 15-6:**

If you were looking at this heading in a live web browser, you'd see that the text is always changing size, making it difficult to ignore.

The way this example works is quite interesting. First of all, you define two variables in the `<head>` section of the underlying HTML. The `size` variable keeps track of the current size of the text (which starts at 10 pixels). The `growIncrement` variable determines how much the text size changes each time your browser runs the code (initially, it grows by two pixels at a time):

```
<!DOCTYPE html>

<html>
<head>
    <title>Dynamic HTML</title>
    <script>
//<![CDATA[
        // The current font size.
        var size = 10
        // The amount the font size is changing.
        var growIncrement = 2
```

Note: This script example is wrapped in a CDATA block (page 430) because it uses the angle brackets to perform a numeric comparison.

Next, the script defines a function named *ChangeFont()*. This function retrieves the HTML object, here the *<p>* element holding the text that will grow and shrink. Once again, the *getElementById()* function does the job:

```
function ChangeFont() {  
    // Find object that represents the paragraph  
    // whose text size you want to change.  
    var paragraph = document.getElementById("animatedParagraph")
```

Now, using the *size* and *growIncrement* variables, you define a variable that performs a calculation to determine the new size for the paragraph:

```
size = size + growIncrement
```

In this example, the *+* performs a numeric addition, because both the *size* and *growIncrement* variables store a number.

It's just as easy to set the new size using the *paragraph.style.fontSize* property. Just tack the letters *px* on the end to indicate that your style setting is measured in pixels:

```
paragraph.style.fontSize = size + "px"
```

If this code runs perpetually, you'll eventually end up with text so ridiculously huge you can't see any of it on the page. To prevent this from happening, you add a safety valve to the code.

Say you decide that, when the text size hits 100 pixels, you want to stop enlarging it and start shrinking it. To do this, you write the script so that it sets the *growIncrement* variable to *-2* when the text size reaches 100. The text starts shrinking from that point on, two pixels at a time. To detect when the message has grown too big, you use conditional logic courtesy of the *if* statement. Here's what it looks like:

```
// Decide whether to reverse direction from  
// growing to shrinking (or vice versa).  
if (size > 100) {  
    paragraph.innerHTML = "This Text is Shrinking"  
    growIncrement = -2  
}
```

Of course, you don't want the shrinking to go on forever, either. So it makes sense to add one last check that determines whether the text has shrunk to 10 pixels or less, in which case the script goes back to enlarging the text by setting *growIncrement* back to 2:

```
if (size < 10) {  
    paragraph.innerHTML = "This Text is Growing"  
    growIncrement = 2  
}
```

Now, here comes the really crafty bit. JavaScript includes a *setTimeout()* function that lets you instruct a browser to "call this function, but wait a bit before you do." In this example, the *setTimeout()* function instructs the browser to call the *ChangeFont()* method again in 100 milliseconds (one-tenth of a second):

```

        setTimeout("ChangeFont()", 100)
    }
//]]>
</script>
</head>

```

Because the ChangeFont() function always uses setTimeout() to call itself again, the shrinking and resizing never stops. However, you can alter this behavior. You could, for example, add conditional logic so that JavaScript calls the setTimeout() method only a certain number of times.

The last detail is the <body> section, which contains the actual paragraph that you resize and a script that calls ChangeFont() for the first time, starting off the whole process:

```

<body>
    <p id="animatedParagraph">This Text is Growing</p>
    <script>
        ChangeFont()
    </script>
</body>

</html>

```

Although the resizing paragraph trick is absurdly impractical, the technique is the basis for many much more impressive scripts (to get the whole script and play around with it, download it from the Missing CD page at www.missingmanuals.com/cds/caw3). For example, you can easily find scripts that animate text in various ways, like making it sparkle, fly in from the side of the page, or appear one letter at a time, typewriter-style.

Events

The most exciting JavaScript-powered pages are *dynamic*, which means they perform various actions as your visitor interacts with them (moving his mouse, typing in text, clicking things, and so on). A dynamic page is far more exciting than an ordinary HTML page, which appears in the browser in one shot and sits there, immobile.

To make dynamic pages, you program them to react to JavaScript *events*. Events are notifications that an HTML element sends out when specific things happen.

For example, JavaScript gives every <a> hyperlink element an event named onmouseover (a compressed version of “on mouse over”). As the name suggests, this event takes place (or *fires*, to use programmer-speak) when a visitor moves his mouse pointer over an HTML element like a paragraph, link, image, table cell, or text box. That action triggers the onmouseover event and your code flies into action.

Here’s an example that displays an alert message when a visitor moves his mouse pointer over a link:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
  <title>JavaScript Test</title>
  <meta http-equiv="Content-Type" content="text/javascript" />
</head>

<body>
  <h1>You Will Be Wowed (Again)</h1>
  <p>When you hover over <a href="SomePage.htm"
    onmouseover="alert('Colorless green ideas sleep furiously.')">this link</a>
  you'll see a secret message.
  </p>
</body>
</html>
```

When you write code to react to an event, you don't absolutely need a script block (although it's a good idea to use one anyway, as shown in the next section). Instead, you can just put your code between quotation marks next to the event attribute:

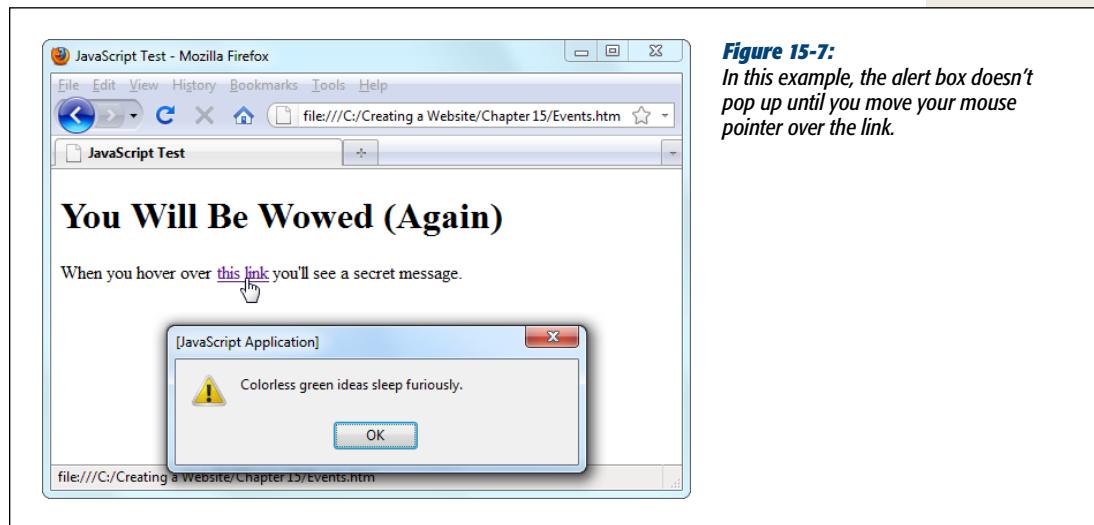
```
<a onmouseover="[Code goes here]">...</a>
```

There's one detail to keep in mind. In this example, the text value ('Colorless green...') uses single quotation marks instead of double quotes. That's because the event attribute itself uses double quotes, and simultaneously using double quotes for two different purposes will horribly confuse your browser.

When you attach code to an event, your browser assumes you're using JavaScript, which is by far the most popular scripting language. However, to meet the strictest rules of HTML, you need to explicitly indicate your language choice by adding the following `<meta>` element to the `<head>` section of your page:

```
<meta http-equiv="Content-Type" content="text/javascript" />
```

Figure 15-7 shows the result of running this script and pointing to the link.



To use events effectively, you need to know what events JavaScript supports. In addition, you need to know which events work on which HTML elements. Table 15-2 provides a list of commonly used events and the HTML elements that they apply to (and you can find a more complete reference at www.w3schools.com/jsref).

In the following sections, you'll learn about two common scenarios that use some of these events.

Table 15-2. Common HTML object events.

Event Name (with uppercase letters)	Description	Applies To Which HTML Elements
onClick	Triggered when you click an element.	Almost all
onMouseOver	Triggered when you move your mouse pointer over an element.	Almost all
onMouseOut	Triggered when you move your mouse pointer away from an element.	Almost all
onKeyDown	Triggered when you press a key.	<select>, <input>, <textarea>, <a>, <button>
onKeyUp	Triggered when you release a pressed key.	<select>, <input>, <textarea>, <a>, <button>
onFocus	Triggered when a control receives focus (in other words, when you position the cursor on the control so you can type something in). Controls include text boxes, checkboxes, and so on—see page 455 to learn more.	<select>, <input>, <textarea>, <a>, <button>
onBlur	Triggered when focus leaves a control.	<select>, <input>, <textarea>, <a>, <button>
onChange	Triggered when you change a value in an input control. In a text box, this event doesn't fire until you move to another control.	<select>, <input type="text">, <textarea>
onSelect	Triggered when you select a portion of text in an input control.	<input type="text">, <textarea>
onError	Triggered when your browser fails to download an image (usually due to an incorrect URL).	
onLoad	Triggered when your browser finishes downloading a new page or finishes loading an object, like an image.	, <body>, <frame>, <frameset>
onUnload	Triggered when a browser unloads a page. (This typically happens after you enter a new URL or when you click a link. It fires just <i>before</i> the browser downloads the new page.)	<body>, <frameset>

Image Rollovers

One of the most popular mouse events is the *image rollover*. To write one, you start by creating an `` element that displays a picture. Then, when a visitor moves her mouse pointer over the image, your browser displays a new picture, thanks to the *onmouseover* event. Creating an image rollover is fairly easy. All you do is get the HTML object for the `` element, and then modify the `src` property.

In this situation, you can't get everything done with a single line of code. You could pile your entire script into the event attribute (using semicolons to separate each line), but the markup would look confusing. A better choice is to write the code as a function. You can then hook the element up to the function using the *event* attribute.

Here's the function to swap an image, for example. In this script, the function is written in a very generic way using parameters, so you can reuse it over and over, as you'll see in a moment. Every time you call the function, you indicate which image you want to change (by supplying the corresponding ID) and what new image file you want to use. Because it uses parameters, you can call the same function to swap any image on a mouse rollover, anywhere on your page.

```
<script>
    function ChangeImage(imageName, newImageFile) {
        // Find the object that represents the img element.
        var image = document.getElementById(imageName)

        // Change the picture.
        image.src = newImageFile
    }
</script>
```

When you create an image rollover, you need to use two events. The *onmouseover* event switches to the rollover picture, and the *onmouseout* event (triggered when your visitor moves her mouse pointer *off* the HTML element) switches back to the original picture. Figure 15-8 shows the result.

```

```

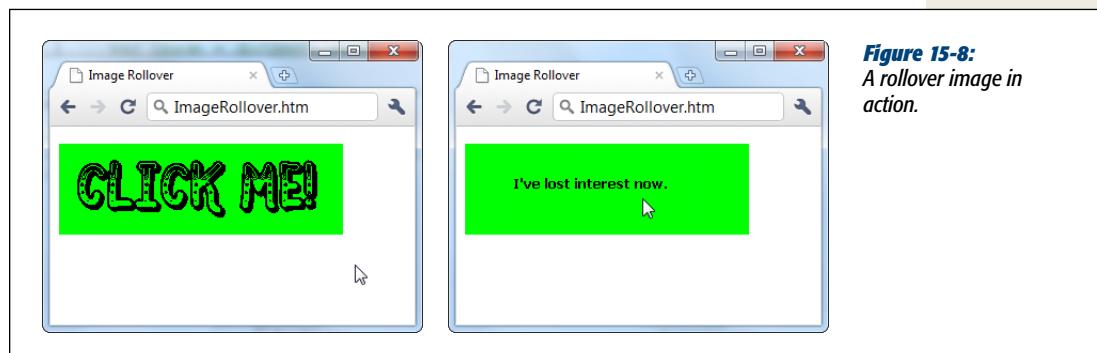


Figure 15-8:
A rollover image in action.

If you want to add more rollover images, just add a new element with a different name. The following element uses the same initial image, but shows a different rollover image each time a visitor moves her mouse on and off the image:

```

```

If you want to get really fancy, you can even use the onclick event (which you trigger when you click an element) to throw yet another picture into the mix.

Note: In Chapter 16, you'll see a different way to approach image rollovers—using CSS. Although the CSS technique doesn't work in every situation, it's a great tool for building basic rollover buttons.

Collapsible Text

Another nifty way to use events is to create *collapsible pages*. The basic idea behind a collapsible page is this: If you've got a lot of information to show your visitors but don't want to overload them with a lengthy page, you hide (or collapse) chunks of text behind headlines that guests can click when they want to see the details (see Figure 15-9).

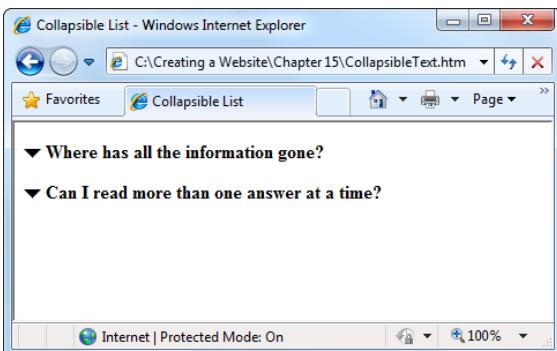
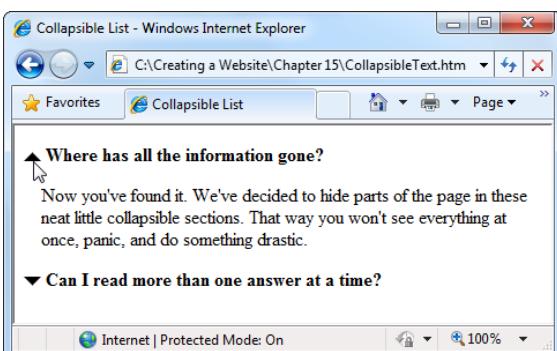


Figure 15-9:
Initially, the browser hides all the body text (top), but when you click the down-arrow image, the browser displays the content for that section (bottom). You can reveal as many sections at a time as you want.



Dynamic HTML gives you many ways to trick browsers into hiding text, and the next example shows one of the best. The technique revolves around the CSS *display* property. When you set this property to *block*, an item appears in the HTML page in the normal way. But when you set it to *none*, the element disappears, along with everything inside it.

The first ingredient in making a collapsible page is to create the function that hides and then shows your content. The function requires two parameters: the name of the open/close image and the name of the element you want to hide or show. The function actually does double duty: It checks the current state of the section, and then it changes that state. In other words, it automatically shows a hidden section and automatically hides a displayed section, thanks to conditional logic. And while the function's doing that, it swaps the open/close image to display a different type of arrow.

Note: This practice, where you always reverse the current state of an item, is called *toggling* by jargon-happy programmers.

```
<script>
    function ToggleVisibility(image, element){
        // Find the image.
        var image = document.getElementById(image)

        // Find the element to hide/unhide.
        var element = document.getElementById(element)

        // Check the element's current state.
        if (element.style.display == "none"){
            // If hidden, unhide it.
            element.style.display = "block"
            image.src = "open.png"
        }
        else
        {
            // If not hidden, hide it.
            element.style.display = "none"
            image.src = "closed.png"
        }
    }
</script>
```

The code starts out by looking up the two objects you need and storing them in the variables *image* and *element*. Then it gets to work. It looks at the current state of the paragraph and makes a decision (using an *if* statement) about whether it needs to show or hide the text. Only one part of this conditional code runs. For example, if a browser is currently hiding the image (that is, if you set the *display* style to *none*), the function runs just these two lines of code, then skips to the bottom of the function and ends:

```
element.style.display = "block"
image.src = "open.png"
```

On the other hand, if the browser is displaying the image, this code gets a chance to prove itself:

```
element.style.display = "none"
image.src = "closed.png"
```

To use this function, you need to add the `` element that performs the toggling on your web page. You also need to add the HTML section that contains the hidden content. You can show or hide virtually any HTML element, but a good all-purpose choice is a `<div>` element because you can stuff whatever you want to hide inside it. Here's an example:

```
<p>

<b>Where has all the information gone?</b>
</p>

<div id="HiddenAnswer1">
<p>Now you've found it. We've decided to hide parts of the
page in these neat little collapsible sections. That way you won't
see everything at once, panic, and do something drastic.</p>
</div>
```

The first part of the markup (between the first `<p>` elements) defines the question heading, which visitors always see. It contains the arrow image and the question (in bold). The second part (in the `<div>` element) is the answer, which your code alternately shows or hides.

Best of all, because you put all the complicated stuff into a function, you can reuse the function to make additional collapsible sections. These sections have the same structure, but different contents:

```
<p>

<b>Can I read more than one answer at a time?</b>
</p>

<div id="HiddenAnswer2" style="display:none">
<p>You can expand as many or as few sections as you want.
Once you've expanded a section, just click again to collapse it back up
out of sight. The only rule is that when you leave this page and come back
later, everthing will be hidden all over again. That's just the way
JavaScript and Dynamic HTML work.</p>
</div>
```

Notice that you have to give each `` and `<div>` element a unique ID or your function won't know which picture to change and which section to hide.

Optionally, you can change this page to give it a different feel but keep the same collapsing behavior. For example, you can make the page easier to use by letting your visitor expand and collapse sections by clicking the heading text (instead of just the image). The easiest way to do this is to pop the image and the bold heading into a `<div>` element, and then add the `onclick` event attribute to it. Here's the change you

need to make:

```
<div onclick="ToggleVisibility('Question1Image','HiddenAnswer1')">
  <p>
    
    <b>Where has all the information gone?</b>
  </p>
</div>
```

You could even underline the heading text so it looks like a link, which lets viewers know something will happen if they click it. Use the *text-decoration* style sheet property to do that (page 158).

Finally, if you want all your collapsible sections to start off as collapsed, you need to add another script that performs this service. Here's the `<script>` block you need, which you can position at the end of your page, just before the closing `</body>` tag:

```
<script>
  // Hide all sections, one by one.
  ToggleVisibility('Question1Image','HiddenAnswer1')
  ToggleVisibility('Question2Image','HiddenAnswer2')
  ...
</script>
```

You could hide your collapsible sections more easily by setting the *display* style property on each `<div>` element with an inline style rule (page 143). However, this approach can cause trouble in the unlikely event that a visitor has turned JavaScript off in his browser. In this situation, every section will remain permanently hidden. By using the code approach shown here, you ensure that JavaScript-challenged browsers will simply display all the content, including the collapsible sections. The page won't be as impressive, but at least nothing goes missing. This technique, of making sure a page works for everyone but gets added benefits where possible, is called *progressive enhancement*.

Note: You'll see more collapsible text effects when you tackle collapsible menus in Chapter 16.

Interactive Forms

HTML forms inhabit a corner of the HTML standard you haven't explored yet. You can use form elements to create the graphical widgets that make up forms, like text boxes, buttons, checkboxes, and lists. Visitors interact with these components, which are commonly called *controls*, to answer questions and provide information.

Figure 15-10 shows an example of an HTML form in action.

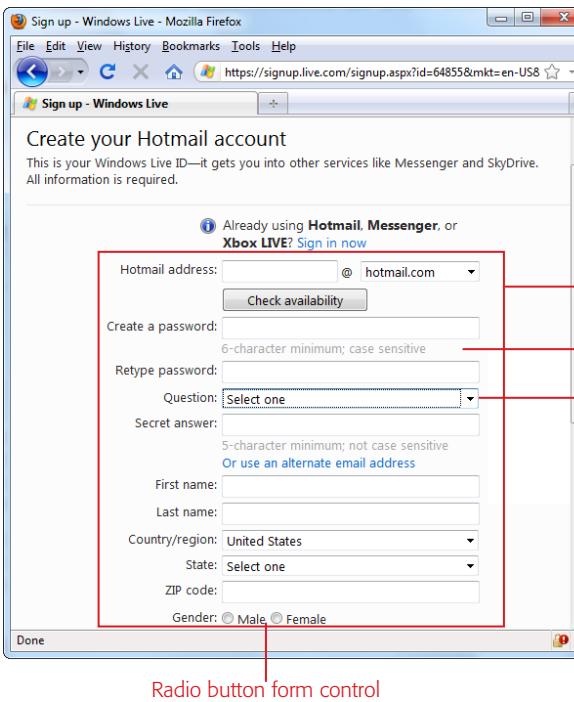


Figure 15-10:
Before Microsoft will grant you a Hotmail account, you need to submit some seriously detailed information. The text boxes, lists, and buttons you see are all part of an HTML form.

HTML forms are an indispensable technology for many websites, but you probably won't get much mileage out of them. That's because they're a hallmark of server-side web applications (page 424). In a typical form, a visitor enters information and clicks a submit button. The browser then collects that information and sends it back to the web server for further processing. This processing might involve storing the information in a database or sending back another page with different HTML (for example, an error message if the application detects a problem).

However, a crafty JavaScript developer can still put a basic form to good use. The difference is that instead of sending information back to a web server, the form collects the data and sends it to a JavaScript routine. That JavaScript code then takes the next step, such as performing a calculation or updating the page.

Form elements

Every HTML form starts out with a `<form>` element. Because `<form>` is a container element, HTML interprets everything inside it as part of your form.

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

Form elements are also block elements (page 107). When you create a form, your browser adds a little bit of space and starts you off on a new line.

What goes inside your `<form>` element? You can put ordinary content (like paragraphs of text) inside or outside of it—it really doesn’t matter. But you should always put form controls (those graphical components, like buttons, text boxes, and lists) *inside* a `<form>` element. Otherwise, you won’t have any way to capture the information a visitor enters.

To create controls, you use yet another set of HTML elements. Here’s the weird part—most form controls use the *exact same* element. That element is named `<input>`, and it represents information you want to get *from* a visitor. You choose the type of input control by using a *type* attribute. For example, to create a checkbox, you use the *checkbox* type:

```
<input type="checkbox" />
```

To create a text box (where a visitor types in whatever text he wants), you use the *text* attribute:

```
<input type="text" />
```

Every `<input>` element also supports a *value* attribute, which you usually use to set the initial state of a control. For example, if you want to put some text inside a text box when a page first appears, you could use this markup:

```
<input type="text" value="Enter the first name here" />
```

Checkboxes are a little different. You can start them off so that they’re turned on by adding the *checked="checked"* attribute, as shown here:

```
<input type="checkbox" checked="checked" />
```

Not all controls use the `<input>` element. In fact, there are two notable exceptions. You use the `<textarea>` element to grab large amounts of text—copy that spans more than one line. (Don’t ask why HTML uses a new element for this purpose—it’s largely for historical reasons.) You use the `<select>` element to create a list from which a visitor selects items. Table 15-3 lists all of the most common controls. It doesn’t include the new form ingredients that have been added in HTML5 (and aren’t usable in Internet Explorer 8 and earlier).

Table 15-3. Form controls.

Control	HTML Element	Description
Single-line text box	<code><input type="text" /></code> <code><input type="password" /></code>	Shows a text box where visitors can type in text. If you use the <code>password</code> type, the browser won’t display the text. Instead, visitors see an asterisk (*) or a bullet (•) in place of each letter as they type in their password, hiding it from prying eyes.
Multiline text box	<code><textarea>...</textarea></code>	Shows a large text box that can fit multiple lines of text.

Control	HTML Element	Description
Checkbox	<code><input type="checkbox" /></code>	Shows a checkbox you can turn on or off.
Radio button	<code><input type="radio" /></code>	Shows a radio button (a circle you can turn on or off). Usually, you have a group of radio buttons next to each other, in which case the visitor can select only one.
Button	<code><input type="submit" /></code> <code><input type="image" /></code> <code><input type="reset" /></code> <code><input type="button" /></code>	Shows the standard clickable button. A <i>submit</i> button always gathers up the form data and sends it to its destination. An <i>image</i> button does the same thing, but lets you display a clickable picture instead of the standard text-on-a-button. A <i>reset</i> button clears the visitor's selections and text from all the input controls. A <i>button</i> button doesn't do anything unless you add some JavaScript code.
List	<code><select>...</select></code>	Shows a list where your visitor can select one or more items. You add an <code><option></code> element for each item in the list.

A basic form

To create a complete form, you mix and match `<input>` elements with ordinary HTML. For example, consider the page shown in Figure 15-11. It provides several text boxes where a visitor types in numbers, and then it uses those numbers to perform a calculation when the guest clicks the Calculate button.

Building this example is surprisingly easy. The trickiest part is creating the function that powers the underlying calculations.

This function needs several pieces of information, corresponding to the values in the three text boxes (feet, inches, and pounds). The function also needs the name of the element where it should display the results. Here's how the function starts:

```
<script>
    function CalculateBMI(feet, inches, pounds, resultElementName) {
```

Note: You could create a `CalculateBMI()` function that doesn't use any parameters. Instead, the function could just search for all the controls on the page by name. However, using parameters is always a good idea, because it makes your code more flexible. Now you can use the `CalculateBMI()` function on all kinds of different pages, with or without a form.

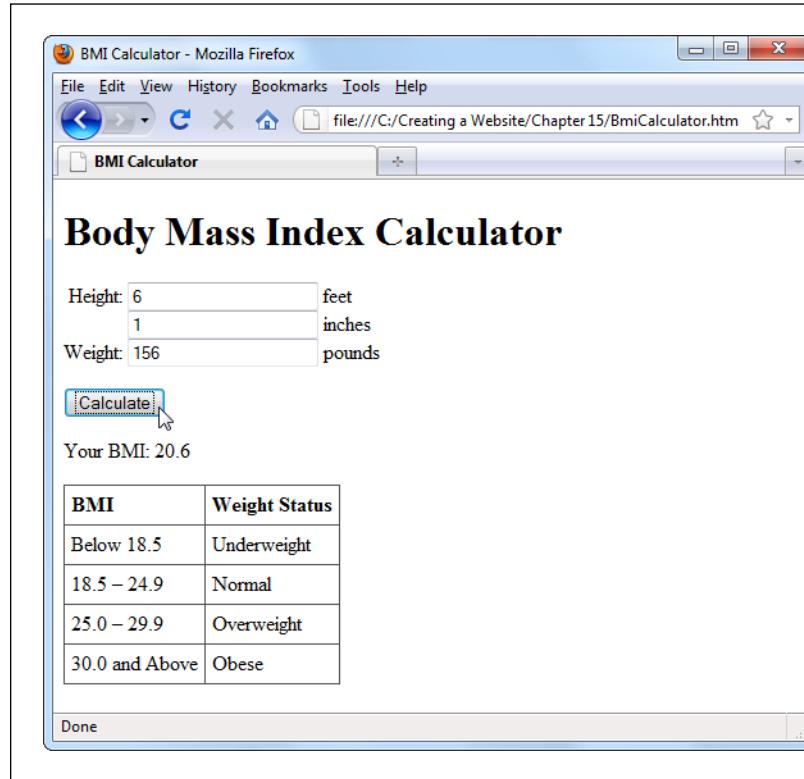


Figure 15-11:
While most visitors are more concerned about what this BMI calculator says about their health, you recognize single-line text boxes and a submit button at work.

The function code that follows isn't much different from what you've seen before. One trick is that it begins by using a *Number()* function that's hardwired into JavaScript. This function converts the text a visitor types in to numbers that the function can use in calculations. If you don't take this step, you might still get the right answer (sometimes), because JavaScript can automatically convert text strings into numbers as needed. However, there's a catch—if you try to *add* two numbers and JavaScript thinks they're strings, it will just join the two strings into one piece of text, so 1+1 would get you 11. This mistake can really scramble your calculations, so it's best to always use the *Number()* function, like so:

```
inches = Number(inches)
pounds = Number(pounds)
feet = Number(feet)
```

The actual calculation isn't too interesting. It's taken straight from the definition of Body Mass Index, which you can find on the Internet.

```
var totalInches = (feet * 12) + inches
```

Finally, the function displays the result:

```
var resultElement = document.getElementById(resultElementName)
resultElement.innerHTML =
    Math.round(pounds * 703 * 10 / totalInches / totalInches) / 10
}
</script>
```

Creating the form that uses this function is the easy part. All you do is create the text boxes with `<input>` elements and give them names you can easily remember. In the example below, the form uses a table to make sure the text boxes line up neatly next to each other:

```
<form action="">
<table>
    <tr>
        <td>Height:&nbsp;</td>
        <td><input type="text" name="feet" /> feet</td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input type="text" name="inches" /> inches</td>
    </tr>
    <tr>
        <td>Weight:&nbsp;</td>
        <td><input type="text" name="pounds" /> pounds</td>
    </tr>
</table>
```

Finally, at the bottom of the form, you create a button that calls the `CalculateBMI()` function using the form's values. To have the button make this call, you need to program your page to react to the `onclick` event. To look up a value in a form, you don't need the `getElementById()` function. Instead, you access it by name through the `this.form` object, which represents the current form:

```
<p>
    <input type="button" name="calc" value="Calculate"
        onclick="CalculateBMI(this.form.feet.value, this.form.inches.value,
        this.form.pounds.value, 'result')"/>
</p>
</form>
```

The final ingredient is the element that displays the result. In this case, because you want it to appear inside another paragraph, the `` element makes more sense than a `<div>` element (see page 117 to review the difference).

```
<p>
    Your BMI: <span id="result"></span>
</p>
```

You can use all sorts of other form-related scripts. For example, you can check the information that people enter into forms for errors before letting them continue from one page to another. To learn more about these scripts, you need to take your search to the Web, as described in the next section.

Scripts on the Web

JavaScript is a truly powerful tool. If you're a die-hard alpha nerd who likes to program your TiVo to talk to your BlackBerry, you'll enjoy long nights of JavaScript coding. However, if you don't like to lie awake wondering what `var howMany = (trueTop > 1?"s" : "")`; really means, you'll probably be happier letting someone else do the heavy lifting.

If you fall into the nonprogrammer camp, this chapter has some good news. The Web is flooded with free JavaScript. Most of the time, these scripts include step-by-step instructions that explain where to put the functions, what elements to use in your page, and how to hook your elements up to functions using events.

However, there's a downside to free JavaScript. As you learned at the beginning of this chapter, JavaScript dates back to the early days of the Internet, and many JavaScript sites are nearly as old. As a result, they may feature garish formatting, out-of-date browser compatibility information (for example, they might warn you that a script doesn't work on the long-deceased Netscape browser), and old approaches that have been replaced with more modern techniques. Some JavaScript sites are also chock full of ads.

If these issues haven't discouraged you, here are some good starting points for your JavaScript search:

- www.dynamicdrive.com

This site provides a set of respectable scripts that emphasize Dynamic HTML. Some scripts create exotic effects, like glowing green letters that tumble down the page, Matrix-style.

- www.huddletogether.com/projects/lightbox2

This site has just a single script, but it's one of the most popular effects on the Web today. If you've ever clicked on a picture thumbnail and had it expand to full size (while the rest of the page goes subtly dark), you've seen a variation of this effect.

- www.javascriptkit.com/cutpastejava.shtml

This site has a pile of scripts from the bad old days of the Internet. There are a few gems, and a lot of free scripts to play with.

- www.webmonkey.com/tutorial/JavaScript_Tutorial

This site offers a small set of useful JavaScript tutorials, which can help you get oriented in the language—and pick up a few core techniques.

Using this list, you can dig up everything from little frills to complete, functioning Tetris clones. But keep in mind that a script is only as good as the coder who created it. Even on sites with good quality control, you could stumble across a script that doesn't work on all browsers or slows your page down to a sluggish crawl. As a rule of thumb, always try out each script thoroughly before you start using it on your site.

Tip: The hallmark of a good script site is that it's easy to navigate. You'll know you've found a bad script site if it's so swamped with ads and pop-ups that you can't find the scripts themselves.

Finding a Simple Script

Ready to hunt for scripts online? The next series of steps takes you through the process from beginning to end.

1. Fire up your browser and choose your site from the list above.

In this example, use www.dynamicdrive.com.

2. Choose the category you want from the site's home page.

In this case, go to the Documents Effects category. For a sample of what else you can find, see the box below.

UP TO SPEED

Script Categories

To get a handle on the types of Dynamic HTML scripts available, look through the different categories at Dynamic Drive (www.dynamicdrive.com). Here's a sampling of the categories you'll find:

- The **Calendars** category scripts produce nifty HTML that looks like calendars—great for displaying important dates or letting visitors plan in advance.
- The **Date & Time** category offers live clocks and countdowns to a specific date.
- The **Document Effects** category provides page transitions and background effects (like fireworks or floating stars).
- The **Dynamic Content** category has menus that slide out, sticky notes, and scrollable panels.
- The **Form Effects** category includes scripts that let you manage forms (see page 453). You can use them to make sure visitors submit forms only once, check for invalid entries, and more.
- The **Games** category offers full-blown miniature games, like tic-tac-toe and Tetris. These games stretch the capabilities of JavaScript and Dynamic HTML as far as they can go.
- The **Image Effects** category has slideshow and image gallery scripts, along with dynamic images that change pictures when you move your mouse.
- The **Links & Tooltips** category includes fancy links that flash, button tricks, and pop-up text boxes that capture your visitors' attention.
- The **Menus & Navigation** category provides handy collapsible menus and navigation bars that let visitors move through your site, like the components you'll see in Chapter 16.
- The **Mouse and Cursor** category offers scripts that change the mouse pointer and add those annoying mouse trails (pictures that follow the mouse pointer wherever it goes).
- The **Scrollers** category has marquee-style scrolling text, like you might see in a news ticker.
- The **Text Animations** category scripts bring text to life, making it shake, fly, glow, or take on even more bizarre characteristics.
- The **User/System Preference** category scripts dig up information about the browser that's currently displaying your page.
- The **Window and Frames** category has scripts for a dozen different types of pop-up windows.

3. Scroll through the list of scripts in your category (Figure 15-12), and then click one.

In this case, use the Top-Down Stripy Curtain Script.

The browser compatibility information
may be your most important consideration

The script's description gives some basic information
Here's a script that sounds interesting

Top-Down Stripy Curtain Script FF1+ IE5+ Opr7+
 A script that opens up a "curtain" (up-down) as the surfer enters the page.

Top-Down Curtain Script FF1+ IE5+ Opr7+
 A script that opens up two "panels" (up-down) as the surfer enters the page.

Left-Right Curtain Script FF1+ IE5+ Opr7+
 A script that opens up two "panels" (left-right) as the surfer enters the page.

Boxing-In Script FF1+ IE5+ Opr7+
 A script that creates a boxing-in effect as the surfer enters the page.

Boxing-away Script FF1+ IE5+ Opr7+
 A script that "boxes away" to reveal the document beneath as the surfer enters the page.

4. The next page shows an example of the script (Figure 15-13).

Once the page loads, you find a script description, the author's name, and a link to the script (if it isn't already displayed on the page). Underneath all this information are the step-by-step instructions for the script.

Top-Down Stripy Curtain Script
 A script that opens up a "curtain" (up-down) as the surfer enters the page. These scripts create effects very similar to ones seen on TV, such as gradually bringing the page into view etc.

Compatibility: FF1+ IE5+ Opr7+

Description: A script that opens up a "curtain" (up-down) as the surfer enters this page. These scripts create effects very similar to ones seen on TV, such as gradually bringing the page into view etc.

Instructions: You just need to enter this page and press the Enter key. As you do, the curtain will open up. Please press the Esc key to close it again.

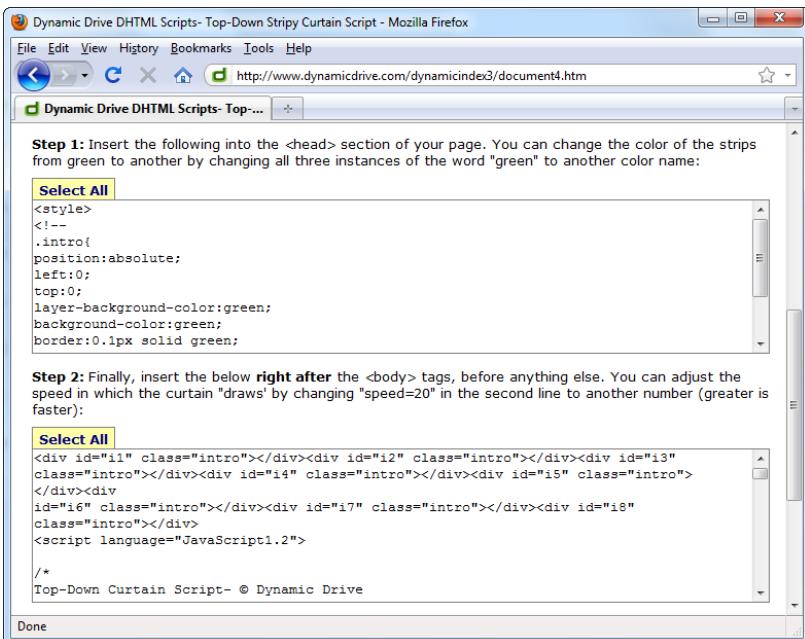
Figure 15-13:

Here's the Top-Down Stripy Curtain Script in action. It fills in the page by drawing alternating strips, some from top to bottom and others from bottom to top. It all happens in a flash.

- Follow the instructions to copy and paste the different parts of the script into your page (Figure 15-14).

You often get a set of functions you need to put in the <head> portion of your page and then some HTML elements you need to place in the <body> section. In some cases, you can customize the scripts—for example, you might modify numbers and other values to tweak the script code, or change the HTML elements to provide different content.

Note: Many scripts include a set of comments with author information. If they do, the standard practice is to keep these comments in your script file, so other developers who check out your site will know where the code came from. This practice is just part of giving credit where credit's due. Ordinary web visitors won't even think to look at the script code, so they won't have any idea whether or not you wrote the script from scratch.



```

Dynamic Drive DHTML Scripts- Top-Down Stripy Curtain Script - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.dynamicdrive.com/dynamicindex3/document4.htm
Dynamic Drive DHTML Scripts- Top...
Select All
<style>
<!--
.intro{
position:absolute;
left:0;
top:0;
layer-background-color:green;
background-color:green;
border:0.1px solid green;
}

Step 1: Insert the following into the <head> section of your page. You can change the color of the strips from green to another by changing all three instances of the word "green" to another color name:
</style>
<!--
.intro{
position:absolute;
left:0;
top:0;
layer-background-color:green;
background-color:green;
border:0.1px solid green;
}

Step 2: Finally, insert the below right after the <body> tags, before anything else. You can adjust the speed in which the curtain "draws" by changing "speed=20" in the second line to another number (greater is
/* Top-Down Curtain Script- © Dynamic Drive
<div id="i1" class="intro"></div><div id="i2" class="intro"></div><div id="i3" class="intro"></div><div id="i4" class="intro"></div><div id="i5" class="intro"></div><div id="i6" class="intro"></div><div id="i7" class="intro"></div><div id="i8" class="intro"></div>
<script language="JavaScript1.2">
/*
Top-Down Curtain Script- © Dynamic Drive
</script>
</body>
</html>

```

Figure 15-14:
The Top-Down Stripy Curtain Script has two components. The first is a style definition that produces the solid-background curtain that the page's content wipes away. The second part creates the background curtain (as a <div> element) and includes the code that performs the transition. Copy both of these components to any page, and you're set. (For even better organization, consider placing the code in a separate JavaScript file, as described on page 439.)

JavaScript Libraries

In the years since JavaScript was first created, development has moved on to JavaScript *libraries*. These libraries go beyond a hodgepodge of individual scripts. Instead, they act as a whole new value-added layer of JavaScript goodness, extending the JavaScript language so you don't need to write everything from scratch.

Even better, JavaScript libraries ensure that the features they offer work the same in all of today's browsers—sometimes using huge reams of behind-the-scenes code to do so. And super-smart programmers use JavaScript libraries to build even more useful scripts, including slideshows, product carousels, dynamic charts, and image magnifiers. These slick scripts go far beyond the examples you'll find on an old-fashioned JavaScript site. For all these reasons, professional web developers almost always use JavaScript libraries.

Here's a list of the most popular JavaScript libraries out there:

- jQuery (<http://jquery.com>)
- MooTools (<http://mootools.net>).
- Prototype (www.prototypejs.org)
- Dojo (<http://dojotoolkit.org>)

If you've never touched a line of programming until this chapter, JavaScript libraries might not be for you. That's because they're designed for other programmers—people who want to create their own JavaScript-fuelled web pages, but don't want to reinvent the wheel. Mere mortals can use them, but there's a steep learning curve. If you're just getting started, you can check out *JavaScript: The Missing Manual*, which describes JavaScript basics and the ins and outs of jQuery.

Web Widgets

A new trend is taking over from cut-and-paste JavaScript sharing, without requiring the complexity of a JavaScript library: web widgets. Essentially, a *web widget* is a self-contained, miniature application that you can pop into one or more web pages. A typical widget fits in a small box on the page, and is powered by JavaScript. Common examples include widgets that show constantly updated news headlines, offer interactive games, play music, display slideshows, poll visitors, collect data with forms, and much more. You can search for a widget to call your own at one of the many widget repositories on the Web, such as www.widgetbox.com, www.google.com/ig/directory?synd=open, and www.widgipedia.com.

The real difference between web widgets and ordinary cut-and-paste JavaScript code is that the widget runs on someone else's web server. That means the widget code you copy is often quite simple. Here's the complete code for a weather forecasting widget from www.widgetbox.com, for example:

```
<script
  src="http://cdn.widgetserver.com/syndication/subscriber/InsertWidget.js">
</script>
<script type="text/javascript">
  if (WIDGETBOX)
    WIDGETBOX.renderWidget('97ec07e5-5918-47da-b3ef-9c4b92013564');
</script>
```

Most of the code is stored on the cdn.widgetserver.com web server, in a file named InsertWidget.js. By running on another server, the JavaScript code can access resources on that server, like real-time information, foreign exchange rates, or (in this case) weather forecasts.

When you run the page, the browser downloads this JavaScript and file runs the code. That JavaScript creates the widget, which then displays itself in your page (Figure 15-15).

Note: You've already seen this approach at work in several places in this book. For example, Google uses JavaScript to fetch a suitable block of ads for your AdSense-enabled web page (page 395). It's also how the Yahoo Media Player plugs its MP3 player into your web pages (page 498).

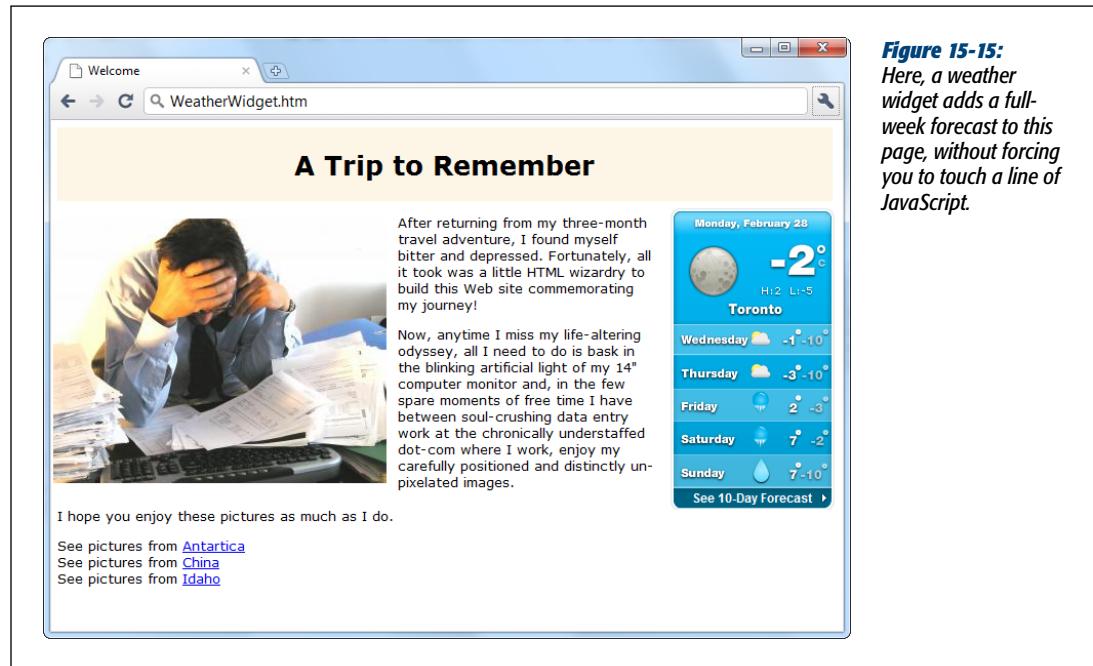


Figure 15-15:
Here, a weather widget adds a full-week forecast to this page, without forcing you to touch a line of JavaScript.

Web widgets are a great tool if you can find exactly what you want. Usually, there are relatively few customization options—for example, you may be able to change the weather forecast city and the size of the weather widget, but you can't change other details about its appearance or behavior. (Some widgets also charge you to get more features, or display tiny ads.) If you need more control, or if you want to design a widget of your own, you need to get much deeper into JavaScript.

Fancy Buttons and Menus

Over the past 15 chapters, you've absorbed a fair amount of web wisdom. You learned to structure web pages using HTML, clothe them with style sheets, and breathe life into them with JavaScript. Now, it's time to reap some of the rewards.

In this chapter, you'll learn how to master three of the most common (and most practical) web design techniques. You'll learn how to build fancy buttons—for example, ones that light up when a mouse hovers over them. You'll also use a pop-up or pop-open menu system, so visitors can cruise around your site in style.

These features give you the chance to take the skills you've developed in CSS and JavaScript one step further. In other words, it's time for your hard slogging to pay off with some snazzy website frills.

Fancy Buttons

The trends and styles of web design are always changing. In the early days of the Web, everyone used ordinary text links, like the ones you learned about in Chapter 8:

```
<a href="graceland.html">Visit Elvis</a>
```

Over time, these run-of-the-mill links started to look drab. Creative webmasters wanted more, and they decided to use small, clickable pictures instead. To do this, they pulled out the familiar `` element and applied one of two techniques.

The most straightforward approach to creating a graphical button is to wrap the image in an anchor, as described in Chapter 8. Here's what that looks like:

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

Unfortunately, when you use this method, HTML adds an ugly blue border around the image to indicate that it's a link. To get rid of the border, you set CSS's *border-style* attribute (see page 172) to *none*.

The second, alternative approach is to use the `` element in conjunction with JavaScript's `onclick` event attribute (page 448). Here's what that looks like:

```

```

This method doesn't generate an ugly blue border, so you don't need to set the *border-style* attribute. However, you do need to create a JavaScript function named `GoToGraceland()` with the code that does what you need. This technique is worth the trouble if you don't want to redirect your guest to a new page, but you do want to run some JavaScript instead. For example, you could perform a calculation, change the content in the other elements on your page, and so on. You used this approach with the BMI calculator in Chapter 15 (page 457)—visitors clicked a button to have the calculator perform an arithmetic operation and display the result.

Pretty soon, web designers weren't happy with text links *or* fancy button pictures. They wanted more, and they used JavaScript to get it. The basic idea was to create a new sort of button that uses the JavaScript image-swapping technique you learned about on page 449. These dynamic buttons (also known as *rollover buttons*) change subtly but noticeably when a visitor hovers her mouse pointer over them. This effect tells her she's poised over a real, live button, and all she needs to do is click it to complete the deal.

For a while, rollover buttons were wildly popular, and virtually everyone used them. And then, leading website designers grew a bit tired of all the whirly, glowy button effects, and they decided to strip their pages down to a leaner, classier look. They reduced their use of rollover buttons and made the rollover effects themselves simpler. For example, web designers might change a button's background color on rollover, but plop ordinary HTML text on top of it. Not only does this create pages that are less busy and less distracting, it makes them easier to maintain, because you don't need to generate dozens of different button pictures. Figure 16-1 shows the typical way that site designers use fancy buttons today.

You already know how to link text and ordinary, unchanging pictures. And you also know the basics of the JavaScript image-swapping technique (it's described on page 449). But what you don't yet know is how to put it all together in a modern package—one that ensures your buttons look understated but cool, one that loads your pictures with no annoying lag, and one that uses clever CSS tricks to keep your website free of messy JavaScript. That's what you'll learn next.

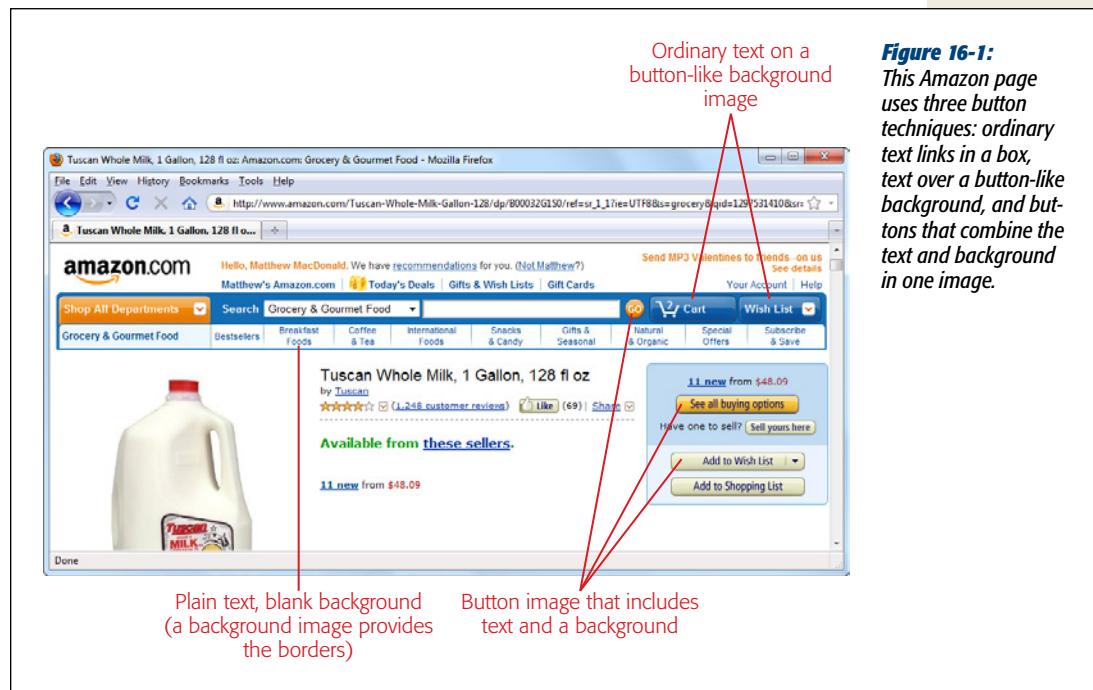


Figure 16-1:
This Amazon page uses three button techniques: ordinary text links in a box, text over a button-like background, and buttons that combine the text and background in one image.

Creating Your Button Pictures

Before you continue, you need to decide what your button should look like, and how you plan to create it.

The first decision to make is whether your button will be a single graphic or whether you'll use ordinary HTML text on top of a background button picture (see Figure 16-2). The first approach gives you the most control and allows the most impressive effects. However, it's also more effort, because it forces you to create a separate pair of pictures for each button. For example, if you want to put eight graphical buttons on your web page, you need 16 button pictures, one for each button's normal state and one for its highlighted (mouse-over) state. But if you use the second approach (which has become the standard on more restrained, business-oriented sites), you need just two pictures: a regular button background and a highlighted button background. You reuse those pictures for every button you create.

Note: Using full button pictures isn't just more effort when you create a website, it's also more effort when you change it. For example, if you need to change the text on a button, you need to regenerate two pictures. If you want to alter the size of your buttons, you need to regenerate all of them. If you use just two background button pictures, you avoid these headaches.

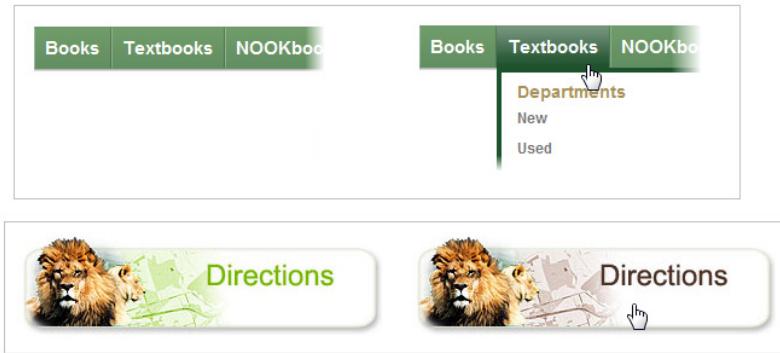


Figure 16-2:
Top: This button uses ordinary text with a green background that darkens when a mouse moves over it.

Bottom: This graphical button swaps pictures on mouseover.

If you decide to go with button pictures, you need a way to create them. Your options are:

- **Draw them yourself.** If you're graphically inclined, you can create button pictures by hand using just about any graphics program (Adobe Photoshop and Adobe Fireworks are two popular choices). However, getting buttons to look good isn't always easy. It's also hard to mass-produce them, because you need to make every button consistent: same text position, size, color palette, and background.
- **Use a button-creation website.** If your artistic abilities are feebler than those of Koko the painting gorilla, there's an easier option. You can use a specialized *button creation* program. These programs have no purpose in life other than to help you create attractive buttons with the text, colors, and backgrounds you choose. The Web teems with a range of these tools. They usually start by asking you to specify button details (like the text, color, background, and so on). Once you finish, you simply click a button and the program creates the button image (or images) and displays them in a new page. All you need to do is download the images and start using them on your site. Two examples of online button-making tools include www.buttongenerator.com and www.grsites.com/button.

Note: Not all button-makers can create images for each button “state” (unclicked, hovered over, clicked, and visited). However, you can usually run the button generator multiple times and choose a slightly different color scheme to create the highlighted button image.

- **Use Expression Web.** The popular web design tool has a feature that lets you create a whole whack of button pictures and the JavaScript code that manages them. The only disadvantage is that this feature relies on a slightly cumbersome JavaScript-based approach, rather than the CSS technique that most web designers now prefer. To try it out, choose Insert→Interactive Button from the Expression Web menu. You get to choose from an impressively featured button generator.

If you decide to use a simple button background with superimposed text, your life is easier. Just about anyone can develop two good-looking rectangles in a drawing program (one for the normal button state, and one for the highlighted state). And because you create just two buttons and no text, you won’t face the headaches of trying to get text to line up correctly in each button (which, if done incorrectly, can create maddeningly inconsistent buttons). If you don’t want to make your own graphics, you can take to the Web and use Google’s image-search tool to find a ready-made button background you want to adopt. Or, you can avoid images altogether and use nothing but CSS properties to create an even simpler button background, as you’ll see on page 474.

In the following sections, you’ll see how to create a page that uses the simplest and most modern approach to buttons: a rollover button that uses a background picture, and manages its states with CSS. You’ll also see how to tweak a page to use the more complicated picture-with-text approach.

Note: There’s another reason to prefer ordinary text superimposed on a button background: search engines prefer it. As you learned on page 289, search engines give special weight to the text inside an anchor link. But if you use the image-only approach, you lose the chance to get that bit of extra attention.

Making a Rollover Effect with CSS

Once you have the button images you need, you’re ready to incorporate them into a web page. Let’s say you want to build the buttons shown in Figure 16-3.

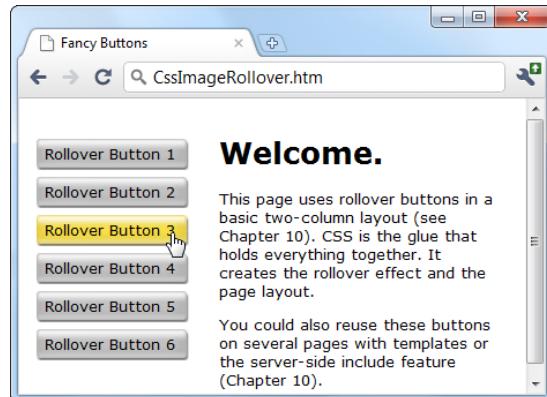
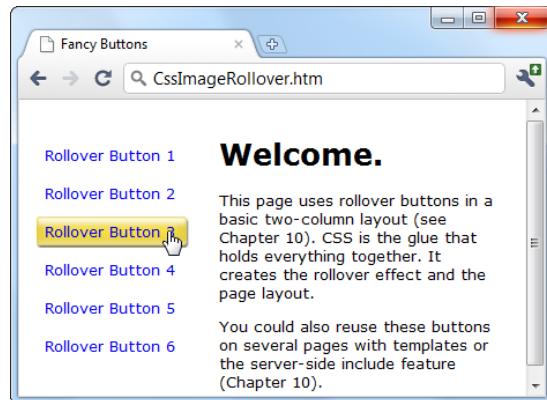


Figure 16-3:
Top: Here, the rollover effects swap a gray shaded background for a yellow one.

Bottom: Sometimes, you can get a more modern look by using an effect that leaves the background initially blank.



Structurally, this is a simple page. It uses two `<div>` elements—one for the list of links on the left, and one for the main content that occupies the rest of the space.

```

<div class="MenuBar">
  <a href="#">Rollover Button 1</a>
  <a href="#">Rollover Button 2</a>
  <a href="#">Rollover Button 3</a>
  <a href="#">Rollover Button 4</a>
  <a href="#">Rollover Button 5</a>
  <a href="#">Rollover Button 6</a>
</div>

<div class="MainContent">
  <h1>Welcome.</h1>
  <p>...</p>
  <p>...</p>
</div>

```

The magic that underpins this page is all in the style sheet. First, apply a basic body rule to the whole page, so you get a consistent font:

```
body {  
    font-family: Verdana, sans-serif;  
    font-size: small;  
}
```

Then, position the menu bar on the left side of the page, using the layout properties you picked up in Chapter 9:

```
.MenuBar {  
    position: absolute;  
    top: 20px;  
    left: 0px;  
    margin: 15px;  
}
```

Give the main content extra margin space on the left, so it can't overlap the menu:

```
.MainContent {  
    margin-left: 165px;  
    margin-top: 30px;  
    margin-right: 20px;  
}
```

So far, these style rules are straightforward. But things get more interesting with the rule that picks out all the anchor elements in the menu bar (using a contextual selector; see page 180):

```
.MenuBar a {  
    display: block;  
  
    text-decoration: none;  
    color: black;  
  
    background: url("NormalButtonBackground.jpg") no-repeat 0 0;  
  
    width: 125px;  
    height: 23px;  
    margin-bottom: 5px;  
    padding-top: 5px;  
    padding-left: 7px;  
}
```

This HTML has several effects on your button. First, the *display* property transforms from an inline element into a block element (like a paragraph or heading). That way, a browser puts each link in the sidebar on a separate line, so you don't need to add line breaks.

Note: Another approach is to make each element a separate item in an unordered list (as represented by the ul element). You can then use style sheets to format the list so that it doesn't show the standard bullet next to each item. Many of the dynamic, pop-up, and collapsible menus you'll come across on the Web are actually unordered lists on the inside.

Next, the *text-decoration* and *color* properties change the text from its standard look (underlined blue) to something that looks more appropriate on a button. But the real magic starts with the *background* property. It grabs the image NormalButtonBackground.jpg and puts it behind each anchor. The no-repeat clause makes sure the browser doesn't tile the image, and the 0 0 values position the picture's top-left corner at the top-left corner of the anchor.

Finally, the *width* and *height* properties size the anchor to match the size of the background button (125 × 23 pixels), while the *margin* and *padding* properties separate the buttons from one another and pad the text inside.

Of course, none of this creates the rollover effect. To get that, you need the *hover pseudo-class* (first described on page 223). It springs into action when a mouse hovers over an element. In this case, it changes the background picture:

```
.MenuBar a:hover {  
    background: url("HighlightedButtonBackground.jpg") no-repeat 0 0;  
}
```

Tip: If you use a less button-like, more box-like background, you might want to set the *text-decoration* property to *underline* in the hover pseudo-class. That way, the link becomes underlined when someone moves a mouse over it, making its page-navigating purpose clear. Amazon uses this trick in its pop-up menus and in the category-browsing bar that sits across the top of some windows (for instance, in Figure 16-1).

This completes the example. However, you may remember that CSS defines a few more pseudo-classes, and you can use those in your button, too. Use *visited* to control what a link looks like once a guest visits the linked page. Use *active* to control what a link looks like in the brief moment when a visitor clicks it. For example, you could shift the button slightly to the side to make it look like it's being pushed in:

```
.MenuBar a:active {  
    margin-left: 1px;  
}
```

Alternatively, you could supply another background image.

Picture-with-Text Buttons

You can adapt the previous example to work with full button pictures, where you don't supply any link text, but you face a few challenges.

First, you need to think about image *preloading*. This ensures that a browser downloads all the button "state" images when it processes the page for the first time, instead of waiting until a visitor moves his mouse over a button. Although you won't notice the difference when you run the page from your computer's hard drive, pre-loading images makes the buttons more responsive when visitors interact with them over the Internet, particularly if they have a slow connection, and particularly if your web page holds a lot of buttons.

Web designers have tried a number of tricks for preloading images. Eventually, they settled on packing both button images (the normal and the selected state) into a single image file. That way, the browser grabs just one image, when it first displays the button. Figure 16-4 shows a sample button. (Incidentally, these images were created at the www.buttongenerator.com website mentioned earlier, and then pasted together into one file.)

**Figure 16-4:**

This graphic combines a normal (top) and a highlighted button image (bottom). Your style sheet rules can grab just one part of the picture and use it as a background. Note that it doesn't matter how much space you put in between the two pictures, as long as you keep the space consistent across all your pictures.

You can expand this double-button system into a triple-button system if you use the *active* pseudo class. And some performance-mad web designers pack a whole page worth of buttons and other graphical embellishments into a single background image. They then use carefully targeted style rules to slice and dice the graphic and spread it around all the elements that need it.

The double-button technique also helps immensely if you create picture-with-text buttons, because it means you'll have fewer files to manage and fewer files to refer to in your markup.

Once you prepare some buttons, you're ready to build an example like the one shown in Figure 16-5.

**Figure 16-5:**

A page with fancy rollover buttons.

The first step in using image-based buttons is to remove all the text from your anchors, so that only the button shows through. Then you need to supply the appropriate full-button images. But here's where things start to get messy. In the previous example, you stored the background pictures using two style rules. But if you want each button to use a different set of pictures, you need a separate style rule for every button—and that's sure to make a mess of even the best-organized style sheet.

A better solution is to use the style to store all the information *except* the button picture file names. Here's the slightly shorter style rule for formatting anchor elements:

```
.MenuBar a {  
    display: block;  
  
    width: 115px;  
    height: 18px;  
  
    margin-bottom: 5px;  
    padding-top: 5px;  
    padding-left: 7px;  
  
    background-repeat: no-repeat;  
}
```

Now, you can specify the figure file for each button using an inline style, like this:

```
<a href="..." style="background-image: url('DogButton.png')"> </a>  
<a href="..." style="background-image: url('CatButton.png')"> </a>  
<a href="..." style="background-image: url('LemurButton.png')"> </a>
```

The style rule explicitly sets the width and the height of the buttons, so there's only room to show the top part of the background button.

The final ingredient is the rule that formats hovered-over links. It offsets the button just a shade (by setting the background-position using a negative number of pixels), effectively pulling the picture up, so the top portion falls outside the top margin of the anchor button and the bottom portion is visible. In Figure 16-5, the highlighted button sits 23 pixels below the normal-state button, so the style sheet rule looks like this:

```
.MenuBar a:hover {  
    background-position: 0 -23px;  
}
```

Picture-less Buttons

Some minimalist web designers build buttons without any pictures at all. They use CSS properties to create a shaded box behind their text. This approach is a bit more modest, and it suffers from one flaw. Browsers won't fully embrace some of the CSS features that really turn an ordinary background into a reasonable facsimile of a button—things like linear gradients, shadows, and rounded borders—until HTML5 takes off. In the meantime, you can get them to work in Firefox, Chrome, Opera, and Safari, but not any version of Internet Explorer before IE 9.

If you want to try using the CSS-only approach, check out the button-maker on the CSS-Tricks site (<http://css-tricks.com/examples/ButtonMaker>). It lets you adjust colors and shading to create a poor man's button (Figure 16-6).

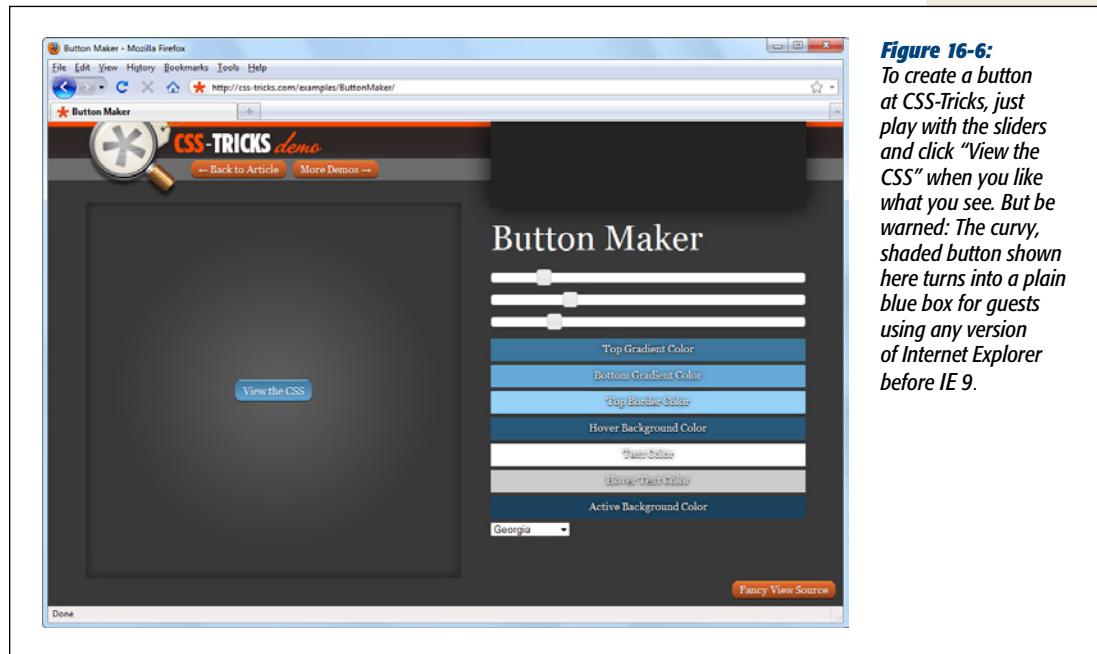


Figure 16-6:
To create a button at CSS-Tricks, just play with the sliders and click "View the CSS" when you like what you see. But be warned: The curvy, shaded button shown here turns into a plain blue box for guests using any version of Internet Explorer before IE 9.

Fancy Menus

When rollover buttons were first created, they were wildly popular. There's something irresistible about a button that lights up when you point to it. You can, however, have too much of a good thing, and stuffing too many rollover buttons on a page is a surefire way to create an overdone turkey of a website.

More recently, the Web's seen a small renaissance in simplicity and a trend away from excessive rollover buttons. Part of the reason is the increasing complexity of websites—quite simply, a handful of rollover buttons no longer offers enough navigational aid for today's typically complex sites. Instead, these sites use more detailed multilevel menus, replacing dozens of rollover buttons with a clearer, more streamlined set of hierarchical links.

A typical menu starts with a collection of anchor elements that you group together on a page. The key is to organize these links into logical groups. For example, the website for a company might include a group of product pages, a group of pages with contact and location information, and another group of tech support pages. By arranging links into groups, visitors can find what they're looking for more easily.

So far, this menu design doesn't require anything special. Using the linking skills you picked up in Chapter 8 and the layout smarts you gained in Chapter 9, you can easily create a side panel with a grouped list of anchors. But really neat menus add another flourish—they're *collapsible*. That means you don't see the whole list of links at once. Initially, you see only the group headings, like Products, Contact Information, and Tech Support. When you click a group heading, a list of related links pops open just underneath.

You can create collapsible menus several ways. Some are easy, while others are dizzyingly complex. In the following sections, you'll learn how to build a simple collapsible menu of your own, and then use a more complicated menu courtesy of a free JavaScript site.

Do-It-Yourself Collapsible Menus

You can create a respectable menu of your own using the collapsible Dynamic HTML trick described in Chapter 15 (page 450). The basic idea is to use JavaScript to hide and show specific HTML elements by changing the CSS *display* property.

Imagine you want to create the cool tabbed menu shown in Figure 16-7. You split the links into three groups, each of which you represent with an on-screen tab. Only one tab shows its sublinks at a time.

This two-level design (the uber links on the page represent the first level, and the links that appear when you hover over a group link represents the second level) might seem a little intimidating, but it consists of only two parts: the tabs at the top of the page and the link boxes (menus) that appear dynamically underneath them when a visitor hovers over the link boxes. To make these regions easy to deal with, you might think to wrap them in `<div>` and `` elements, as you've seen throughout this book.

Note: In the rest of this section, you'll get a chance to look at the solution piece by piece. To see the complete page, check out the downloadable content for this chapter, available from the Missing CD page at www.missingmanuals.com/cds/caw3.

Because the three tabs appear next to each other on the same line, the `` element is the best choice. Remember, the `<div>` element adds a line break and some space between each element. The `` element is an inline element, which means you can fit it inside an existing paragraph and put more than one `` element side by side.



Here's the HTML that represents the three panels:

```
<div class="TabGroup">
  <span class="Tab">About Me</span>
  <span class="Tab">My Store</span>
  <span class="Tab">Really Cool Stuff</span>
</div>
```

Figure 16-7:

Top: When this page first loads, it presents visitors with three tabs.

Middle and bottom: As a visitor moves her mouse over a tab box, a set of related links appears underneath. These links "float" above the page content.

These `` elements have the descriptive class name `Tab`. That associates them with the following style sheet rule, which gives the tabs the correct font and borders:

```
.Tab {  
    font-weight: bold;  
    padding: 5px;  
    border-style: solid;  
    border-width: 1px;  
    cursor: hand;  
}
```

This style includes something you haven't seen yet—the `cursor` property. It styles the mouse pointer that appears when a guest hovers her mouse over a link element. In this case, the mouse pointer changes to a hand icon (Figure 16-7, middle).

You wrap all the tabs in a `<div>` that uses the `TabGroup` class. That lets you put the `TabGroup` at a specific position on the page:

```
.TabGroup {  
    position: absolute;  
    top: 16px;  
    left: 10px;  
}
```

After you declare the tabs, it makes sense to add the floating submenus. Each submenu is simply a box with borders and a yellow background. Inside, you find a group of links. The `<div>` element makes sense here, because you want each submenu to exist independently on the page (rather than stuffed together into a single line). You also need to give each `<div>` element a unique ID, so you can change its visibility based on the tab a visitor clicks.

Here are the `<div>` elements for the three link groups:

```
<div id="AboutMe" class="Links">  
    <a href="...">My Traumatic Childhood</a>  
    <a href="...">My Education</a>  
    <a href="...">Painful Episodes</a>  
</div>  
<div id="MyStore" class="Links">  
    <a href="...">Buy Something</a>  
    <a href="...">Request a Refund</a>  
    <a href="...">File a Complaint</a>  
</div>  
<div id="ReallyCoolStuff" class="Links">  
    <a href="...">Pie Eating</a>  
    <a href="...">Harvesting Bananas</a>  
    <a href="...">Blindfolded Heart Surgery</a>  
</div>
```

The `<div>` elements float above the page, which means you need to absolute position them. Here's the style rule for that:

```
.Links {  
    position: absolute;  
    top: 40px;  
    left: 10px;
```

```

border-width: 1px;
border-style: solid;
padding: 10px;
background-color: lightyellow;
font-size: small;

display: none;
}

```

Along with the absolute positioning coordinates (40 pixels from the top of the browser window, 10 pixels from the left), this style also sets a few a few formatting details (the border, background, padding, and text size). More importantly, it uses the *display* property to explicitly hide all the submenus when the page first loads. So even though this example stacks the submenu `<div>` elements one on top of the other, you won't ever see them that way on a page, because you won't ever see them all at once.

Give the `<a>` elements inside the floating boxes a bit of margin space so they don't run into each other:

```

.links a {
    margin-right: 5px;
}

```

Tip: If you want a menu that stacks its links one above the other, you can tweak this style rule to use the *display: block*, just as you did with the panel of rollover buttons on page 471.

And, finally, wrap the rest of the content for the page in a `<div>` element with the class name MainBody. Give this `<div>` a generous top margin, so that it clears the tabs:

```

.MainBody {
    margin-top: 70px;
    margin-left: 15px;
}

```

These style sheet rules and `` and `<div>` elements create the basic framework for your page. The final step is to create a script that displays one of the hidden `<div>` elements, depending on which tab your visitor selects.

The code that does this is similar to that you used in the `ToggleVisibility()` function in Chapter 15 (page 453). But in this case, you're not interested in hiding and showing individual sections; you want to show a single section, depending on the tab selected, and hide everything else. Two custom functions handle the job: `MakeVisible()`, which shows the submenu for a specific tab; and `ResetAllMenus()`, which hides all the submenus.

Here's a simplified version of the `MakeVisible()` function. As you can see, it takes an element name, finds the element, and changes its style settings so that it appears on the page.

```
function MakeVisible(element){
    // Find the element and unhide it.
    var element = document.getElementById(element)
    element.style.display = "block"
}
```

Now you can hook this function up to all the tab buttons. You have a choice here: MakeVisible() could react to either a click using the onclick event or to a mouse pointer hovering over the tab using the onmouseover event. This example uses the latter approach.

```
<span class="Tab" onmouseover="MakeVisible('AboutMe')">About Me</span>
<span class="Tab" onmouseover="MakeVisible('MyStore')">My Store</span>
<span class="Tab" onmouseover="MakeVisible('ReallyCoolStuff')">Really Cool
Stuff</span>
```

The page still isn't quite right. Although the MakeVisible() function shows the correct tabs, it doesn't hide anything. That means that if you pass your mouse over all three tabs, you see all three groups of links at the same time, one above the other.

To hide the irrelevant tabs, you need to get a little craftier. The problem is that MakeVisible() knows what tab it's supposed to show, but it doesn't know the status of the other two tabs. To find that out, your code needs to search through the rest of the page. In this example, the basic approach is to look for any <div> element that has the class name Links and hide it. The ResetAllMenus() function handles that:

```
function ResetAllMenus() {
    // Get an array with div elements.
    var links = document.getElementsByTagName("div")

    // Search the array for link boxes, and hide them.
    for (var j = 0; j < links.length; j++) {
        if (links[j].className == 'Links') links[j].style.display = "none"
    }
}
```

This code is a little tricky. First, the getElementsByTagName() function retrieves a programming object called an *array*. An array is special because it doesn't hold just one object, it holds a whole group of objects at once. In this case, the array named *links* holds three objects, one for each <div> element on the page.

Then you use a programming construct called a *for* loop. It processes code a certain number of times using a built-in counter. In this case, the counter is a variable named *j* that starts at 0 and keeps increasing until it matches *links.length*—in other words, until it gets to the last <div> object in the *links* array. Assuming the *links* array has three items, your browser executes this statement three times:

```
if (links[j].className == 'Links') links[j].style.display = "none"
```

The first time, *j* is 0, and the code loads up the first object in the list. The second time, *j* is 1, and it digs up the second object. You can guess what happens the third time. As the code moves through this list, it checks the class name of each <div> element. If it indicates that you found a link box, the code makes it disappear from the page by changing its display style.

Note: If the stranger aspects of JavaScript still look like Danish, don't worry. If you're inclined, you can learn about JavaScript programming features like arrays, loops, and if statements from a website or dedicated book (like *JavaScript: The Missing Manual*). Or, you can keep your sanity and rely on the examples provided with this book and find great free scripts online.

Now you can fix the MakeVisible() function so that it first hides all the menus, and then reveals just the one you want:

```
function MakeVisible(element){  
    ResetAllMenus()  
  
    // Find the element and unhide it.  
    var element = document.getElementById(element)  
    element.style.display = "block"  
}
```

A good practice is to hide all the floating menu boxes if your guest moves his mouse pointer off the floating link box and over the rest of the page. This suggests that the visitor decided not to click a menu command and went back to reading the page:

```
<div class="MainBody" onmouseover="ResetAllMenus()">
```

The code in the downloadable example gets slightly fancier. It hides a selected tab's border and changes its background color. However, the basic approach is the same.

Third-Party Menus

If you've had enough fun writing your own JavaScript code, you'll be happy to hear that the Web is chock full of free menu scripts. Many of these have more dazzle than the tabbed menu shown in the previous example. Some of the extra features you might find include:

- Multilevel menus that let your visitors drill down to specific subcategories
- Menus that let you collapse and expand subsections, so you can show all the links that interest you
- Ridiculously showy effects, like shaded highlighting and transparent backgrounds

To find a good menu, you can use a JavaScript sample site (page 459), or search for “JavaScript menus” or “CSS menus” on the Web. You'll find that there's quite a bit more diversity in menus than in rollover buttons. Every menu looks and behaves a little differently. Some pop up, others slide out, and others try to emulate the look and feel of popular programs like Microsoft Outlook.

Tip: Stay away from menus that force you to bury your links in a block of JavaScript code. Not only is this approach less maintainable, it can cause problems for search engines, which might not be able to discover (and index) all the pages on your website. Instead, a modern menu should put the links in real <a> elements, which you can then stack one after the other or place inside an unordered list.

To get a glimpse of what's out there, head over to Dynamic Drive, which has a nifty set of menus at www.dynamicdrive.com/dynamicindex1 and a particularly interesting specimen called the Slashdot menu at www.dynamicdrive.com/dynamicindex1/slashdot.htm. Figure 16-8 shows this menu with the same structure as the tabbed menu you saw earlier in this chapter.

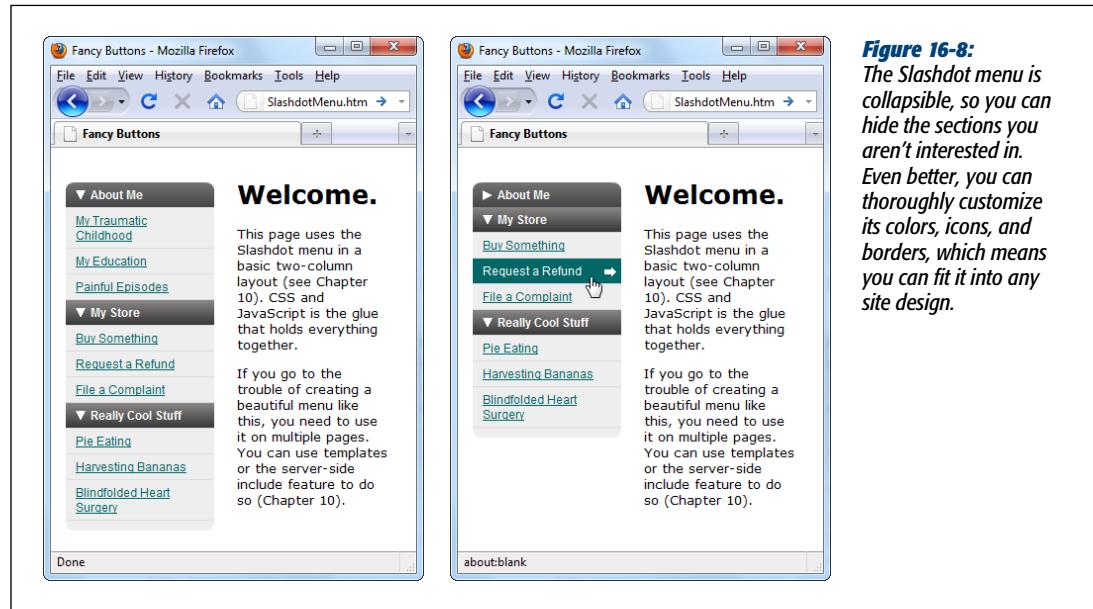


Figure 16-8:
The Slashdot menu is collapsible, so you can hide the sections you aren't interested in. Even better, you can thoroughly customize its colors, icons, and borders, which means you can fit it into any site design.

Tip: Before you choose a navigation bar for your own site, test drive quite a few. This section walks you through the process, but you'll want to compare the results with other navigation bars before you commit to one.

In the following sections, you'll download the script for a Slashdot menu and put it to use.

Getting the script

To download the Slashdot menu, follow these steps:

1. Go to www.dynamicdrive.com/dynamicindex1/slashdot.htm.

This page displays a sample menu and provides instructions for using it. It also describes browser compatibility for the menu, and the news is good—it works in every mainstream browser.

2. Look for the download link for the *sdmenu.zip* file. Click it, and then save the ZIP file somewhere on your computer.

To use the Slashdot menu, you need a JavaScript file, some images, and a style sheet. To make life easy, the ZIP file includes them all.

3. Unzip the contents of *sdmenu.zip*. Put it in your site folder on your computer, along with the rest of your site pages.

All together, *sdmenu.zip* contains a sample page that includes the menu (*index.html*) and a subfolder that contains all the support files (*sdmenu*). Drag both of these into your site folder.

You don't need to touch the contents of the *sdmenu* folder (although you might if you want to refine the menu to match your website). By carefully replacing some of the graphics, you can modify the background of the title sections (*top-title.gif* and *title.gif*), the arrows (*collapsed.gif*, *expanded.gif*, and *linkarrow.gif*), and the bottom border (*bottom.gif*). By cracking open the style sheet (*sdmenu.css*), you can change the background colors, borders, and spacing for the rest of the menu. Finally, the *sdmenu* folder includes a JavaScript file (*sdmenu.js*), which you probably won't edit at all.

4. Create a new web page (or start editing one that already has a set of menu links you want to adapt into a Slashdot menu).

You could edit your *index.html* page (and that's a good way to get started with most examples). However, the Slashdot menu is straightforward and similar to the do-it-yourself collapsible menu you created earlier, so it's easy enough to incorporate into a new page.

Creating the menu

The first step to using the Slashdot menu is to attach its style sheet to your page and add a reference to the JavaScript file that powers the menu. You also need a scrap of script that creates your menu when the page loads. All three of these ingredients go in the *<head>* section of your page, and here's what they look like:

```
<head>
  <title>Fancy Buttons</title>

  <link rel="stylesheet" type="text/css" href="sdmenu/sdmenu.css" />
  <script src="sdmenu/sdmenu.js"></script>

  <script>
    var myMenu;
    window.onload = function() {
      myMenu = new SDMenu("my_menu");
      myMenu.init();
    };
  </script>
  ...
</head>
```

The script code is generic. You can copy it word-for-word into every page that uses the Slashdot menu. The only point to note is that the menu name it uses (*my_menu* in this example) must match the ID of the *<div>* element that contains the Slashdot menu on your page.

You probably also want to add an embedded style sheet or a link to another style sheet in your <head> section. The example in Figure 16-8 uses three basic style rules. One assigns a default font to the page, another positions the sidebar that has the Slashdot menu, and the third positions the main content section:

```
body {
    font-family: Verdana, sans-serif;
    font-size: small;
}

.MenuBar {
    position: absolute;
    top: 20px;
    left: 0px;
    margin: 15px;
}

.MainContent
{
    margin-left: 180px;
    margin-top: 30px;
    margin-right: 20px;
}
```

These styles are nothing new. You saw them in earlier examples.

Now it's time to build the menu. Its structure is remarkably similar to the examples you've seen in this chapter. Essentially, each collapsible section of the menu consists of a <div> container full of anchor elements. The only added feature is the title text, which a element at the top of the <div> provides:

```
<div>
<span>About Me</span>
<a href="...">My Traumatic Childhood</a>
<a href="...">My Education</a>
<a href="...">Painful Episodes</a>
</div>
```

A typical Slashdot menu contains several collapsible subsections. You wrap them all together in another <div> element, and give it a name that matches the menu name in the script:

```
<div class="sdmenu" id="my_menu">
...
</div>
```

This is enough to create the Slashdot menu with all its formatting and functionality intact. However, you probably want to wrap the Slashdot <div> in another <div>, one that represents the menu sidebar. That way, you can place the sidebar exactly where you want it, without worrying about style sheet conflicts or modifying the *sdmenu.css* file.

```
<div class="MenuBar">
<div class="sdmenu" id="my_menu">
...
</div>
</div>
```

Here's the complete markup used to create the menu in Figure 16-8:

```
<div class="MenuBar">
  <div class="sdmenu" id="my_menu">
    <div>
      <span>About Me</span>
      <a href="...">My Traumatic Childhood</a>
      <a href="...">My Education</a>
      <a href="...">Painful Episodes</a>
    </div>
    <div>
      <span>My Store</span>
      <a href="...">Buy Something</a>
      <a href="...">Request a Refund</a>
      <a href="...">File a Complaint</a>
    </div>
    <div>
      <span>Really Cool Stuff</span>
      <a href="...">Pie Eating</a>
      <a href="...">Harvesting Bananas</a>
      <a href="...">Blindfolded Heart Surgery</a>
    </div>
  </div>
</div>
```

Note: Once you perfect your website and you're ready to take it live, remember to upload the sdmenu subfolder and all its files.

Audio and Video

In the early days of the Internet, websites were about as jazzy as an IRS form. You'd see pages filled with an assortment of plain text, links, and more plain text. Over time, the Web matured, and web pages started to change as designers embraced the joys of color, pictures, and tacky clip-art. But when that excitement started to wear off, it was time for a new trick—multimedia.

Multimedia is a catchall term for a variety of technologies and file types, all of which have dramatically different computer requirements and pose different web design challenges. Multimedia includes everything from the irritating jingle that plays in the background of your best friend's home page to the wildly popular video clip of a cat playing the piano. (Depressing fact: That cat has over 20 million views, you're unlikely to ever create a web page that's half as popular.)

In this chapter, you'll consider how to use several types of multimedia. First, you'll learn to play background music and put a snazzy MP3 player on a web page. Then you'll embed video—first using ordinary HTML, and then using the new, but not-quite-ready `<video>` element in HTML5. Finally, you'll see how you can avoid some serious headaches by hosting your video files on YouTube.

Note: Before you go any further, take a moment to consider the worst examples of multimedia abuse. These include flashing banner ads, irritating background music, time-wasting intro pages, and bandwidth-sucking commercials. Before you jump on the multimedia bandwagon, think about what you want to do. Are you planning to showcase your musical compositions or provide downloadable recordings of Junior's first moments? If so, multimedia probably makes sense. But if you're just looking for a way to dazzle visitors with an animated logo, think twice. It's probably not worth the considerable effort to design something that will only aggravate most of your readers.

Understanding Multimedia

There comes a point when all new web designers want more than mere text and pictures on their pages. Even spruced-up fonts and elegant page layouts don't satisfy the design envy many newcomers feel when they spot a site loaded with sound and motion. That's understandable: You, too, want to trick out your pages with audio and video. But before you can jazz up your site, you need to understand a few basics.

Linking, Embedding, and Hosting

One of the key choices you make when you outfit your pages with multimedia is whether to link to or embed the files.

Linking to multimedia content is the simplest but least glamorous approach. The link *points to* an audio or video file stored along with all your other HTML pages and files. There's really nothing to creating linked multimedia. You use the same lowly anchor element and *href* attribute you used in Chapter 8. Here's an example:

Would you like to hear Industrial Noise?

Figure 17-1 shows what happens when you click one of these babies.

Note: It makes absolutely no difference what kind of software your web host's server runs when you add audio to your site. When someone clicks a link to an audio file, the browser downloads the file to the visitor's computer and plays it there, not from the server.

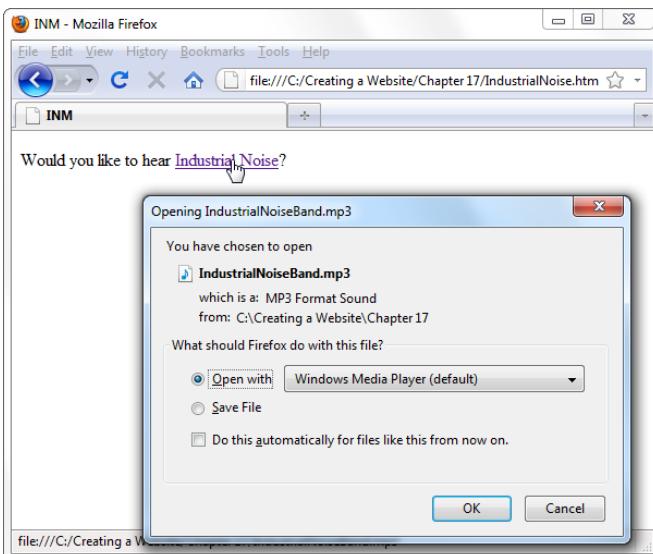


Figure 17-1:

When you click a link to a multimedia file, your browser asks whether you want to save the file or open it straightaway. If you choose the latter, your browser first downloads the file, and then launches it using a separate program. The actual program your browser uses to play the file depends on the software installed on your computer. For example, it might be Windows Media Player or Apple's QuickTime Player.

Embedding multimedia is a more advanced approach. It *integrates* music or video into your HTML page. As a result, you can create rich combinations of text, sound, and video.

But embedding multimedia can be a challenge. That's because HTML lacks direct support for audio and video. Instead, browsers rely on plug-ins (small programs that extend a browser's capabilities). Unfortunately, you have no way of knowing *which* plug-ins your visitors have and (by extension) which audio and video formats their browsers can play back. The bottom line? There's no guarantee that your visitors can see any particular type of multimedia you embed on a page.

If you still insist on embedded multimedia, you'll need to use HTML's slightly disreputable `<embed>` element; you'll learn how it works on page 492. And on page 506, you'll learn about a solution that's just around the corner: HTML5, which introduces a proper `<video>` element. The only catch is that the browser companies are still haggling over which video formats to support.

Note: The distinction between linking and embedding multimedia is the same as the distinction between linking to a picture (with the `<a>` element), and embedding it right in your page (with the `` element). The only difference is that images are a basic, well-supported part of the HTML standard, so embedding pictures never causes much concern. Embedding audio and video, however, takes you into less-wellcharted waters.

But there's one other option for managing multimedia: *hosted multimedia*—multimedia files stored on someone else's server but displayed (or linked to) on your web page. The best-known example of hosted multimedia is YouTube, a ridiculously popular site that plays back hundreds of millions of video clips every day.

Hosted multimedia is an excellent choice if you want to display really large files, particularly movie clips. It won't tap out your website's bandwidth (page 64), and it works with virtually all browsers and operating systems. Its only drawback is that you give up a fair bit of control. For example, if you host your videos on YouTube, you can't show movies that are longer than 15 minutes, and YouTube ratchets down your movie's quality to make sure it performs well. (Technically, YouTube reduces the video's file size so browsers can download them more quickly—that way, visitors experience no delay in playback when they push the play button.) But YouTube is still far and away the most practical way to share clips. It even integrates with your website, so you can put a YouTube window on any web page to show your own videos. You'll learn to use YouTube on page 507.

Types of Multimedia Files

Your decision to link or embed files depends, at least in part, on the type of multimedia you want to showcase. You have a slightly bewildering field of choices:

- **Synthesized music (MIDI).** MIDI files store notes that your computer generates when it plays back the song, rather than playing back a recording of a musical

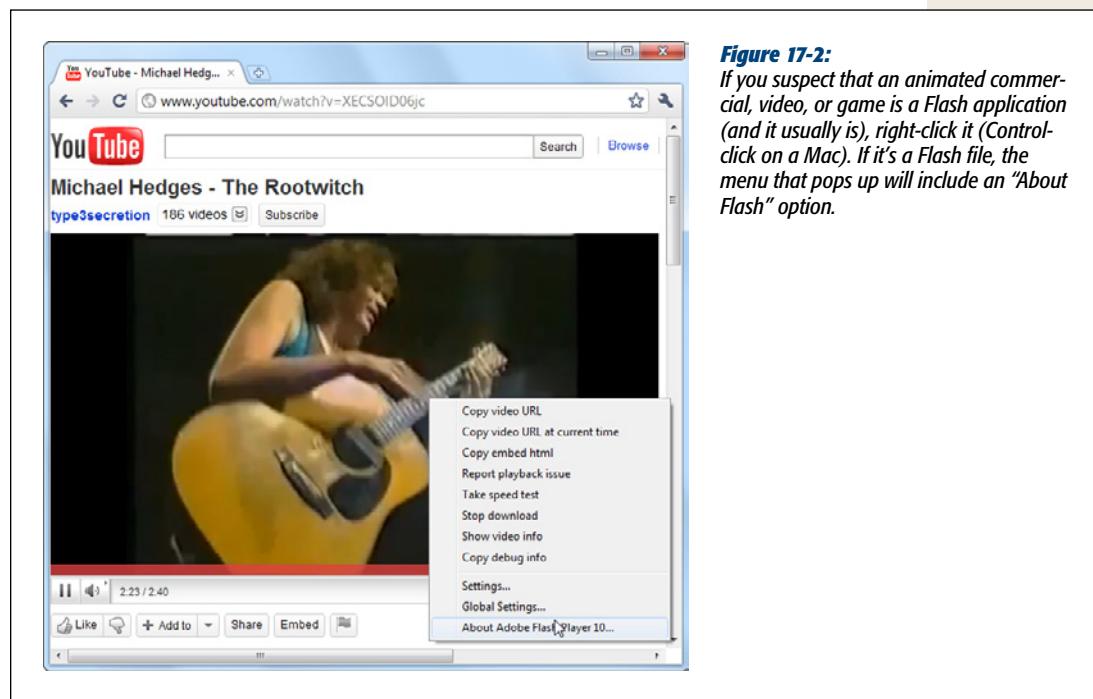
instrument. As a result, MIDI files are small but of questionable quality. Unless you want the effect of a cheesy Casio keyboard, you probably won't use them.

- **Digital audio (WAV and MP3).** These file types store recorded audio, which means they're of higher quality than MIDI files. But WAV files are enormous, making them unsuitable for all but the most bloated websites. MP3 files are one-tenth the size of WAVs, but some browsers won't have the plug-in you need to play them.
- **Digital video (MPEG, AVI, MOV, and WMV).** These file types are multimedia's heavy hitters. They let you play back video that ranges in quality from thumbnail-sized windows with jerky playback to DVD-quality movies. Raw digital video files are a challenge for any web page creator because they're huge. To make digital video play smoothly, you may need to compress, shrink, and reduce your clip's size and quality using video-editing software.
- **Animated GIFs.** Animated GIFs consist of a series of small, still images displayed one after the other in rapid succession, like a flipbook. In the bad old days of the Internet, animated GIFs were everywhere, but now they're fading into history, and few web-heads are sorry to see them go.
- **Flash.** Flash is a versatile plug-in that has a dual identity. On the one hand, it's a tool for programmers to create rich, interactive web applications (like the games you see at www.ferryhalim.com/orisinal). At the same time, Flash is a way to play video (encoded as FLV files) on virtually every type of computing device except Apple iPads and iPhones (Apple doesn't support the Flash standard on mobile devices).

Despite these impressive pluses, Flash has three drawbacks: First, to create a Flash application or Flash video player you need special software from Adobe, which runs into the hundreds of dollars. Second, even if you shell out the Flash cash, there's a learning curve involved. You'll need a helping hand to create a basic video player, and some serious expertise to create the real Flash application. Finally, visitors won't be able to see Flash movies unless they have a Flash plug-in installed. (That said, good estimates suggest that an overwhelming 99 percent of web-connected desktop computers already have the plug-in.)

Note: Multimedia hosts, like YouTube, use Flash to play back their movies (Figure 17-2). That's because Flash gives the best combination of customizability, performance, and compatibility. Of course, these high-powered companies also have plenty of cash to pay their programming teams.

It's difficult to digest all this information at once. If you're still mulling over your choices, take a look at the scenarios in Table 17-1 to help you sort out the roles different types of multimedia play.

**Figure 17-2:**

If you suspect that an animated commercial, video, or game is a Flash application (and it usually is), right-click it (Control-click on a Mac). If it's a Flash file, the menu that pops up will include an "About Flash" option.

Table 17-1. Multimedia scenarios.

If You Want To:	Then Use:	Embedded, Linked, or Hosted
Play a short loop of digital audio continuously in the background	Flash (you can use the MP3 format instead, but not all browsers support it, and the looping is less precise).	Embedded
Let visitors download your band's newest indie recordings	MP3 files. (Record your music using WAV files, then convert them to the MP3 format to save space.)	Linked
Let visitors see your favorite home movie	MPEG, AVI, WMV, or MOV files. (Make sure you use video-editing software to reduce the file size if it's too big.)	Hosted (on a service like YouTube)
Show an animated intro screen, commercial, or story	Flash.	Embedded

Note: If you plan to create a website with a lot of digital audio and video, you need to reconsider the site's storage and bandwidth requirements (see page 65). Unlike ordinary HTML pages and web graphics, multimedia files can grow quite large, threatening to overwhelm your web host's space and bandwidth allotment. You can avoid this problem by using a hosted multimedia service like YouTube, in which case the video views cost you neither web space nor bandwidth.

Basic Background Music

Most people like to browse the Web in peaceful silence. That means no trance-hypno-ambient background tracks, no strange disco beats, and no sudden cymbal crashes. This aversion to noise may be due to the fact that something like 98 percent of all web browsing takes place on company time.

But if you like to startle and annoy people, or if you're absolutely convinced that your web audience really *does* want some funky beats, keep reading to bring on the background music.

The <embed> Element

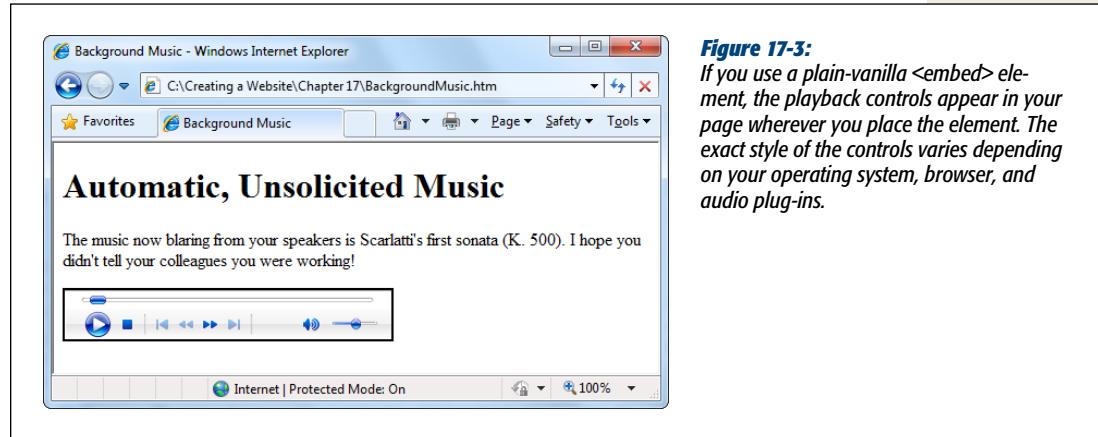
Although the HTML standard doesn't support background music, almost all browsers support the <embed> element, first pioneered by Netscape in the early days of the Web. You can put the <embed> element anywhere on your page. Here's a basic page that uses it to play background music:

```
<!DOCTYPE html>

<html>
<head>
  <title>Background Music</title>
</head>
<body>
  <h1>Automatic, Unsolicited Music</h1>
  <p>The music now blaring from your speakers is Scarlatti's first sonata (K. 500). I hope you didn't tell your colleagues you were working!</p>
  <embed src="scarlatti.mp3" type="audio/mpeg" />
</body>
</html>
```

The <embed> element gives you a slew of options for playback control. If you use the element without specifying any of them (as in the previous example), your visitors see a page like the one shown in Figure 17-3 and hear its audio file automatically.

Music playback isn't always this seamless, however. Because every browser handles embedded music a little differently, you can run into problems like the ones shown in Figure 17-4. The best advice is to test your page on at least the three main browsers (Internet Explorer, Firefox, and Chrome), and consider adding some text to your page that explains the problems guests might encounter.

**Figure 17-3:**

If you use a plain-vanilla <embed> element, the playback controls appear in your page wherever you place the element. The exact style of the controls varies depending on your operating system, browser, and audio plug-ins.

Two screenshots of web browsers demonstrating issues with playing background music. The top screenshot shows Internet Explorer with a security warning message: "This website wants to run the following add-on: 'Windows Media Player from 'Microsoft Corporation''. If you trust the website and the add-on and want to allow it to run, click here...". The bottom screenshot shows Mozilla Firefox with a message: "Additional plugins are required to display all the media on this page." and a link "Install Missing Plugins...". Both screenshots show the same content as Figure 17-3, but the browsers are unable to render the audio due to security or plugin requirements.

Figure 17-4:

Top: Paranoid browsers can lock out your music, depending on their security settings.

Bottom: Depending on what a web visitor has installed or uninstalled, a browser might not find the components it needs to play your background music.

Embedded Audio Options

Ordinarily, the `<embed>` element starts playing music as soon as your browser downloads the specified file. Visitors can kill the sound with a quick click of the stop button (assuming you display the playback controls), but if they're not expecting to hear a burst of music, it's enough to frazzle some nerves.

A more polite way to handle background audio is to display the playback controls and let your visitors decide if and when to click the play button. This design is easy—just use the `autoplay` attribute:

```
<p>If you'd like some soft music to browse by, click the play button.</p>
<embed src="scarlatti.mp3" type="audio/mpeg" autoplay="false" />
```

Turning off autoplay is considered good web etiquette. A much poorer idea is the `hidden` attribute, which lets you hide the playback controls altogether. All too often, you'll find web pages that use `<embed>` elements like this:

```
<embed src="scarlatti.mp3" type="audio/mpeg" hidden="true" />
```

In this example, the sound file plays automatically. And because you hid the playback controls, the only way someone can stop the music is to lunge for the volume control. Websites that put their visitors through this ordeal rarely see a return visit.

The `<embed>` element includes a few more options. Table 17-2 has the lowdown.

Table 17-2. Attributes for the `<embed>` element.

Attribute	Description
<code>src</code>	The URL that points to an audio file.
<code>autoplay</code>	A true or false value that indicates whether the audio should start playing immediately (true) or wait for your visitor to click the play button (false).
<code>hidden</code>	A true or false value that indicates whether the playback controls are visible.
<code>loop</code>	A true or false value that indicates whether the audio should be played once (from start to finish) or repeated endlessly. When looping audio, you'll notice a distinct pause before the audio restarts.
<code>volume</code>	A value between 1 and 100 that specifies playback volume as a percentage of maximum volume. 100 percent is the loudest you can get. 50 percent tends to produce the standard volume on a Windows computer; on Macs, you get that effect at 75 percent. If you set your volume to 100 percent, you can be sure you won't get any repeat visitors. When you use the <code>volume</code> attribute, supply a number only (leave out the % sign).

Sadly, the `<embed>` element won't help you create those nifty looping soundtracks you may have heard on some websites. Even though `<embed>` supports a `loop` attribute, the results aren't good because the song pauses each time you reach the end of the audio file. If you want a slick looping soundtrack, you need to use Flash, as described in the next section.

Tip: Do you need to convert your MP3 files? Lots of great shareware can help you record WAV files and convert them into the more compact MP3 format. Two bargain-basement choices that are free to try are GoldWave (www.goldwave.com) and FlexiMusic (www.fleximusic.com). If all you want to do is convert existing WAV files to MP3 format, you can use Apple's iTunes software, available free for both Windows and the Mac (www.apple.com/itunes). You can get the job done by right-clicking (Control-clicking on a Mac) any song name and choosing "Create MP3 Version" from the pop-up menu.

Flash MP3 Players

As you already learned, Flash is a browser plug-in that lets you add videos, animation, and even whole miniature programs, like games, to a web page. Although it takes a fair amount of work (and some pricey software) to create a Flash program from scratch, it's not nearly as difficult to add a Flash-based music player to your page. That's because plenty of people have already done the work for you. In fact, the Web is awash in free Flash music players.

Search Google for "flash mp3 player" to find a few. Most of them are surprisingly polished, with support for song lists, playback buttons, and customizable colors and themes. In the following sections, you'll see how to use two good choices.

The Premiumbeat Player: Playing a Song List

The www.premiumbeat.com website offers a handful of professional-looking Flash media players. They include:

- **A mini player.** It looks like a play button paired with a tiny bar graph icon (which animates while playback is underway). Use this player if space is at a premium.
- **A thin player.** It looks like the mini player, with the addition of a long slim strip that lets you see where you are in the current track. This is another choice for pages that don't want to draw attention to their music.
- **A basic player.** This includes a full set of playback controls, including next/previous track buttons and a volume slider. Visually, it's a small box that looks like the playback controls in Figure 17-3, but you can optionally show the song name and timestamp.
- **A player with playlist.** This looks like the basic player, but with a list of all the tracks you queued up to play. Use this player if you have more than one piece of music to offer. You can see it in Figure 17-5.

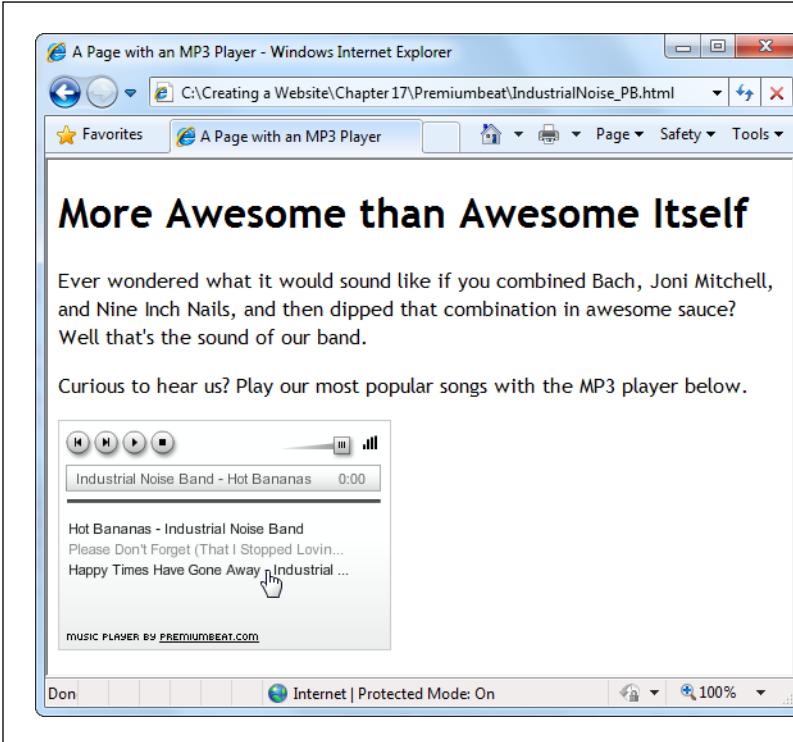


Figure 17-5:
Hit the play button, and this player goes through its playlist from start to finish. Or, double-click a specific song to hear it.

Each player comes in a free version and a premium version. If you pay for the premium version, you can remove the small text “Music Player by Premiumbeat.com” from the bottom of the player.

To use one of the Premiumbeat players, follow these instructions:

1. Visit www.premiumbeat.com/flash_music_players. Find the player you like, and click the nearby Download Free Version button.

This downloads a ZIP file with a small set of files. The .swf file is the actual Flash music player. You can’t open it directly, but you will need to upload it to your website so your pages can use it. The name of the file depends on the exact version of the player you downloaded. If you download the full player with playlist support, for example, you’ll see a file named OriginalMusicPlayerPlaylist.swf.

2. Create a playlist.

A playlist lists all the files you want to load into the player, in order.

The ZIP download includes a sample playlist named *playlist.xml*. You can edit this file or create a new one with a different file name.

Technically, the playlist is an *XML file*, which means it uses tags, just like HTML does. (But unlike HTML, these tags are there to identify different pieces of information for the music player, not to get a document ready for display in a web

browser.) The structure of the playlist is actually quite simple. HTML wraps the entire list in a `<playlist>` element and includes one `<item>` element for each song. Each `<item>` element, in turn, includes a `<title>`, `<artist>`, and `<path>` element. The player displays the `<title>` and `<artist>` information during playback, while the `<path>` element points to the MP3 file you want to hear.

Here's a complete playlist with information for three songs:

```
<playlist>
  <item>
    <title>Hot Bananas</title>
    <artist>Industrial Noise Band</artist>
    <path>HotBananas.mp3</path>
  </item>

  <item>
    <title>Please Don't Forget (That I Stopped Loving You)</title>
    <artist>Industrial Noise Band</artist>
    <path>LoveSong.mp3</path>
  </item>

  <item>
    <title>Happy Times Have Gone Away</title>
    <artist>Industrial Noise Band</artist>
    <path>HappyTimes.mp3</path>
  </item>
</playlist>
```

Tip: The `<path>` element uses the same relative link system you used earlier for anchors and images (page 217). That means the link is always relative to the location of the HTML page that displays the player. To better organize your site, put all your songs into a subfolder (say, named MP3), and then change each song path in your player (for example, from `HotBananas.mp3` to `MP3/HotBananas.mp3`).

3. Add the player to your page.

All you need is the right `<object>` element. The easiest way to get it is to copy the content from the file `sampleEmbedObject.html`, which is included in the ZIP download. The `<object>` element looks like this:

```
<object type="application/x-shockwave-flash" id="player1"
  allowscriptaccess="always" allowfullscreen="true"
  data="OriginalMusicPlayerPlaylist.swf" width="260" height="180">
  <param name="movie" value="OriginalMusicPlayerPlaylist.swf" />
  <param name="FlashVars" value="playlistXmlPath=playlist.xml" />
</object>
```

The only detail you need to worry about is the name of the playlist file (bolded above). If you give your playlist a different name, change the value here accordingly. Optionally, you can tweak the height and width properties too, so that the player fits snugly into your layout.

You'll notice two other sample pages in the Premiumbeat download. The sampleEmbedJavascript.html page creates the player using JavaScript code. It's a neat trick, but only useful if you're integrating your player into some sort of web application. The sampleEmbedMultiplePlayers.html shows how you can put more than one player on the same page, although this is likely to confuse your visitors.

4. Try it out.

Now, when you open your web page in your browser, you should see the media player with your playlist. Click each song to make sure all the links work.

When you're ready to upload the finished product, remember to include your web page, the MP3 files, the XML playlist file, and the .swf media player file (like OriginalMusicPlayerPlaylist.swf).

Tip: If you want to customize your player's font and colors, visit www.premiumbeat.com/flash_player_customization_tool. You can pick the options you want, and the customization tool creates the markup you need to paste into your web page. But if you want to perform a more radical facelift, search for a skinnable Flash MP3 player. Skinnable players let you use custom graphics for every player detail, including the background and playback buttons.

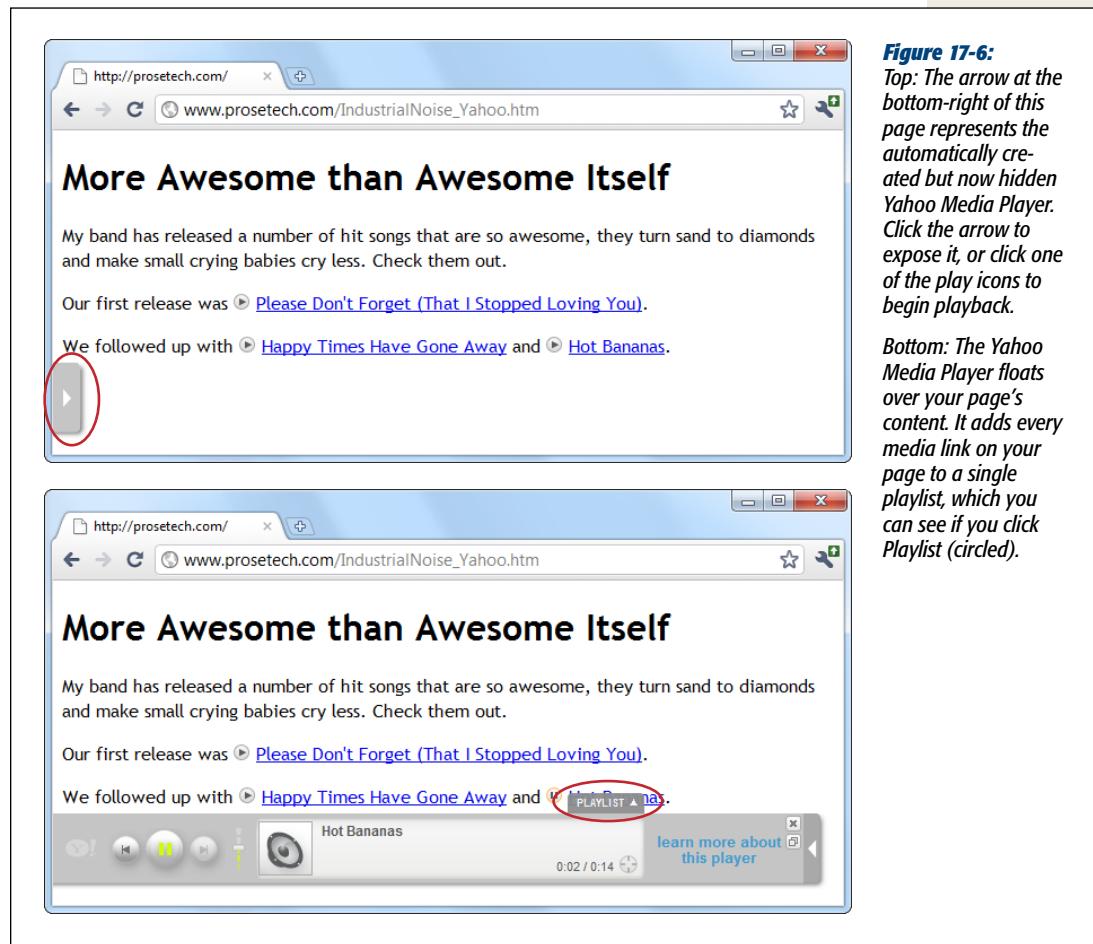
The Yahoo Player: Playing All Your Links

Another way to integrate a player into your pages is with the Yahoo Media Player (<http://mediaplayer.yahoo.com>). It's particularly appealing for time-pressed (or lazy) web designers, because it requires virtually no work. In fact, you don't need to download a single file, because the code for the player resides on Yahoo's web servers. You simply add a single script reference in your web page, like this:

```
<script src="http://mediaplayer.yahoo.com/js"></script>
```

As you learned on page 439, a script reference lets your web page use the JavaScript code in another file. It saves you from cutting and pasting huge chunks of code into every page. The JavaScript code for the Yahoo Media Player is particularly clever. It reacts to web page events. As soon as the browser loads your page, the JavaScript code scans it looking for ordinary <a> links that point to supported media files, like MP3s. Each time it finds a linked MP3 file, it adds a tiny playback icon next to the link. (Of course, the entire process happens in mere microseconds.) Figure 17-6, top, shows the result.

The neat part is what happens when you click one of these playback icons. Instead of prompting you to download the file or attempting to load a plug-in, Yahoo pops up a floating Flash music player (Figure 17-6, bottom). The song begins playing in the floating window, with no annoying security warnings.



Here's the markup for the page in Figure 17-6. As you can see, it's just an ordinary page with hyperlinks. Only the presence of the `<script>` element makes the media player magic happen.

```

<!DOCTYPE html>
<html>

<head>
  <title>A Page with the Yahoo Media Player</title>
  <style>...</style>
  <script src="http://mediaplayer.yahoo.com/js" />
</head>

<body>
  <h1>More Awesome than Awesome Itself</h1>
  <p>My band has released a number of hit songs that are so awesome, they turn sand to diamonds and make small crying babies cry less. Check them out.</p>

```

```
<p>Our first release was <a href="song.mp3">Please Don't Forget (That I  
Stopped Loving You)</a>.  
<p>We followed up with <a href="soundfile.mp3">Happy Times Have Gone  
Away</a> and <a href="song.mp3">Hot Bananas</a>.  
</body>  
<html>
```

Be aware of two caveats. First, the Yahoo Media Player is relatively new and suffers the occasional quirk. (For example, every once in a while the pop-up player may disappear, even though the play buttons next to each link keep working.) Second, it works only if you upload your site to a real, live web server. If you attempt to open a local copy of a page that uses the player (that's a page stored on your computer), it still loads itself, but it won't play anything. The problem is that the MP3 files are trapped on your hard drive, where the Yahoo Media Player can't access them.

Flashtak Loops

In the previous sections, you saw two good ways to add an MP3 player to any web page. But sometimes, you want something simpler. Rather than give your visitors the ability to shuffle through a collection of songs, you might just want to keep them happy with endlessly looping background music.

Although many websites sell audio loops, you can download free ones at Flash Kit, www.flashkit.com/loops (see Figure 17-7). Flash Kit offers a large and excellent catalogue of nearly 10,000 loops ranging in style from ambient to urban.

Note: Loops are the audio equivalent of a wallpaper tile. They're short snippets of music specially designed so that the beginning picks up where the end leaves off. You can play an audio loop over and over again, and the result is a seamless background track. In a first-rate loop, the repetition isn't immediately obvious, and you can happily listen to it for several minutes.

If you download one of these loops as an MP3 file, you can use it with a free Flash MP3 player, like the ones discussed above. But there's another alternative, one that uses a slimmed-down audio format called Flashtak. Flashtak files download in a jiffy, so your visitors never have to wait to experience your site's ambience. They require a Flashtak player, which you can also download for free at the Flash Kit website. (The Flashtak player is a Flash program, just like the Premiumbeat and Yahoo music players, but it doesn't play MP3 files.)

To download the Flashtak player, look for the "Get flashtak players" link just under the links for loops (see Figure 17-7). You can choose from more than a dozen player styles. Most have snazzy effects as they play music, like pulsing lines or expanding circles.

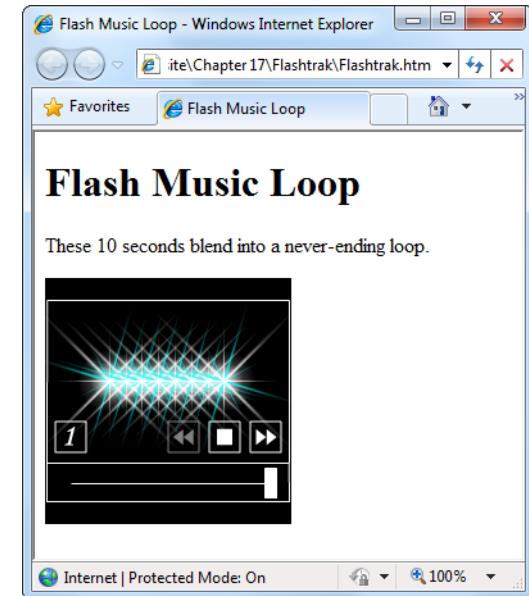
Figure 17-7:
You can preview Flash Kit's loops right in your browser, without downloading them. Once you find what you want, click the "flashtak" link to download the audio in one of three formats: MP3 (the usual), WAV (good if you want to edit it), or in Flashtak format, which works with the specialized players available on the site. In this example, a high-quality 10-second WAV file weighs in at almost 2 MB, but the MP3 version is a more respectable 700 KB. The Flashtak format is even smaller, requiring just 200 KB.

Note: When you download the Flashtak player, you may end up with more files than you actually need. For example, you don't need and can delete any files that end with .fla (these are Flash source files that you can only edit in the Flash software). Also, when you download a player, you'll probably find yourself with a pile of extra song files. Delete the ones you don't want, or your player will cycle through all of them.

Once you download a player, you're ready to embed it in your web page. You can take care of that with a simple <embed> element that points to the player file:

```
<embed src="StarPlayerMultiTrackWithAutoStart.swf"></embed>
```

Figure 17-8 shows you what you'll see when you run the page that contains this element.

**Figure 17-8:**

Here are the Flash-based audio controls in action, complete with playback buttons and soothing graphics. The best way to try out this example (and the MP3 player examples shown earlier) is to download the sample content for this chapter, from the Missing CD page at www.missingmanuals.com/cds/caw3.

Video Clips

Now that you've conquered the challenges of audio and learned to put everything from sound effects to looping background music into your web pages, you're ready to move on to one more challenge—video content.

Although browsers use many of the same tools to play video as they do to play audio (plug-ins like Windows Media Player, QuickTime, and Flash), there are some hefty differences. Most importantly, video files are big. Even the smallest of them is many times the size of an audio recording of a full-length Mahler symphony. Handling this data without trying your visitors' patience is a true test. In the following sections, you'll learn how to prepare your video for the Web and let your visitors view it.

Preparing Video

Putting personal video on a website is a task meant for ambitious multimedia mavens. The key stumbling block is the sheer size of digital video. On a digital camcorder, every second of video can chew through 1 to 3 MB (depending on the recording quality and format you choose). Put together a 10-minute clip, and you're looking at a staggering 600 MB to 1800 MB file. Not only is this awkward to manage, it's enough to take a bite out of any webmaster's server and bandwidth allocations.

What can you do to make a web video both look good and perform well? You can always use someone else's web-ready video (or pay a video-editing company lots of

money to trim yours down to web proportions). Assuming that's not what you want, you have two choices.

- **Record at lower quality.** Many video cameras let you record using lower-quality settings for the sole purpose of putting video on the Web. Cellphones, webcams, mini-camcorders, and digital still cameras all create low-quality movies, letting you dodge conversion headaches and send video straight to your site.
- **Lower the quality afterward.** More commonly, you need to go through a long process of *re-encoding* your high-quality video to convert it to a size suitable for the Web. To do this, you need a video-editing program. Video cameras generally include some sort of tool to help you out, although you may want to pony up for more powerful software. Two basic choices are Windows Movie Maker, included with Windows, and iMovie for the Mac. Many video-editing programs have a feature that automatically picks scaled-down quality settings for videos you want to upload.

Here are the steps to follow to get your video web-ready:

1. **Film your movie.**

Take a couple of lessons from video aficionados and film your video in a way that makes it easier to compress and introduces less distortion: Keep camera movements smooth and gradual, and don't film complex patterns. Your compressed video will look better.

2. **Fire up the video capture program included with your video camera. Use it to download your movie to your computer's hard drive.**

Typically, this step involves connecting your camera to your computer using a *FireWire* cable. Although USB cables aren't fast enough to keep up with huge chunks of raw video data, you might use one if you transfer video from less powerful devices, like a camera or cellphone that records short clips.

3. **Use a video-editing program to snip out just the video segment you want to post.**

Some programs let you add music or special effects at this point, too.

4. **Re-encode that piece of video in a highly compressed format. If all the format information in your program sounds like gobbledegook, look for an option that clearly says "web video" when you save your clip.**

Technically, you make three choices in this step: You specify a video format (the standard your editing program uses to encode your video), the dimensions of the playback window (say, 640×360 for a typical window on a web page), and the video quality (as with JPEGs, the greater the compression, the more detail you lose).

Re-encoding video is a time-consuming operation—even the speediest computer can take five times as long as the length of the original clip. The good news is that at the end of the process, you'll have a more manageable web-ready video file—say, a 20 MB file for a full three-minute clip.

Linking to and Embedding Video

Surprisingly, you can pop a video into your web page using the same techniques you used with digital audio (see Figure 17-1). That means you can link to a video so that it opens up in a new browser window:

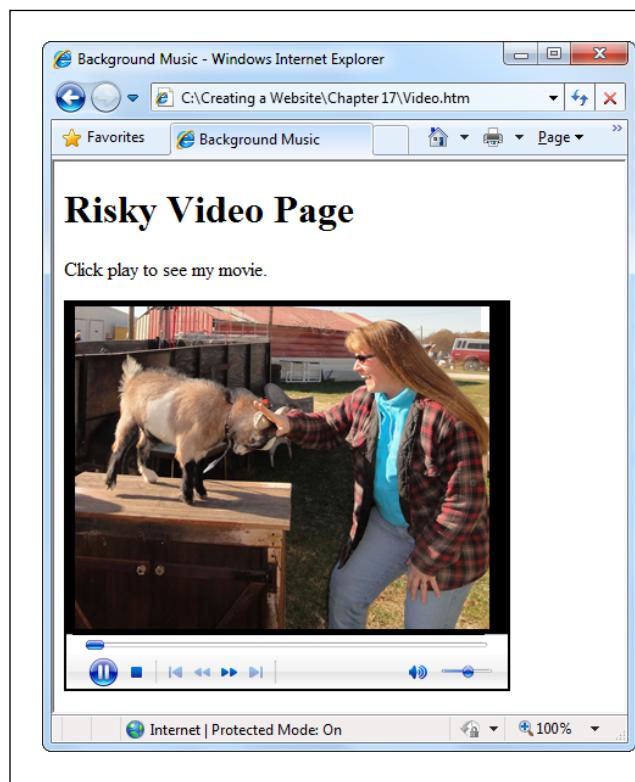
```
Click to download or open my home movie  
<a href="ouch.mpg">Ouch, That Hurts</a>.
```

Or, you can use the `<embed>` element to put a video window right inside your own web page.

```
<embed src="ouch.mpg" autoplay="false" />
```

If you use the `<embed>` element, make sure you turn *off* autoplay. Videos that automatically start playing are even more annoying than auto-start music tracks.

The video window shows up wherever you place the `<embed>` element (see Figure 17-9).



A screenshot of a Windows Internet Explorer browser window titled "Background Music - Windows Internet Explorer". The address bar shows "C:\Creating a Website\Chapter 17\Video.htm". The main content area displays a video player with the title "Risky Video Page" and the instruction "Click play to see my movie.". Below the video player is a video frame showing a woman in a plaid shirt petting a goat. At the bottom of the video player is a control bar with standard media controls (play, stop, forward/backward, volume, etc.). The status bar at the bottom of the browser window indicates "Internet | Protected Mode: On" and "100%". To the right of the browser window, there is a caption and description.

Figure 17-9:
You can add a video window to your web page almost as easily as you can add basic audio playback. If you don't specify a fixed size, the window automatically adjusts to the dimensions of your video.

If this seems too easy to be true, that's because it is. Although this simple test page works well most of the time, it's not entirely reliable. Depending on your movie's encoding format and your browser's settings, visitors may be forced to download an entire movie before they can start watching it. And if their browsers don't have the right plug-in, or if it's incorrectly installed, your video might not play at all.

In the next two sections, you'll consider two solutions that require a fair bit of heavy lifting: Flash and HTML5. Then, you'll consider a nearly perfect alternative with YouTube.

Flash Video

Right now, the most common solution to video plug-in woes is Flash. In fact, the great majority of the video you find scattered across the Web uses Flash. This means two things: The video is encoded in the FLV (Flash video) format, and the video player is a small Flash application, which different websites customize in different ways.

One way to get reliable video on the Web is to use Flash on your web pages. Unfortunately, Flash isn't HTML, and using it requires a whole new set of tools (and a whole new learning process). Odds are, you'll only take this route if you have something else to gain. For example, if you want to create a slick, customized video player, or if you want to use some of Flash's other features, like its ability to make cartoon-like animated stories, arcade games, or other types of graphically dazzling applications. To get started, pick up a dedicated Flash book, like *Flash CS5: The Missing Manual*.

A second option is to use a ready-made Flash video player, which is similar to the Flash MP3 players you used earlier. Two video choices are Flowplayer (<http://flowplayer.org>) and the Premiumbeat video player (www.premiumbeat.com/flash_video_players). Both are free in their basic versions. Unfortunately, Flash video players can play Flash video in the FLV format only, so you also need a video conversion tool. You can find some for free on shareware sites like www.download.com; just search for "FLV encoder."

If both of these options sound like too much hassle, you're a perfect candidate for a video hosting service like the insanely popular YouTube. You'll learn how to use it on page 507.

GEM IN THE ROUGH

Video with Microsoft Silverlight

There's one more plug-in-based solution to web video—you can use Flash's mortal enemy, Microsoft Silverlight.

In the great plug-in wars, Silverlight is the underdog because its first version didn't appear until 2007, long after Flash had colonized the Web. For that reason, Microsoft is unnaturally generous with Silverlight tools. You can download their free Expression Encoder package at <http://tinyurl.com/243b7sr>, which not only converts your video to a compressed, web-friendly format, it also creates a slick Silverlight-powered video player page. (If you're an ambi-

tious sort, you can even customize the video player to get a unique look for your website.)

The only catch is Silverlight's adoption rate. Although it's grown dramatically in the past few years, Silverlight still tops out at about 60 percent of web-connected computers, whereas Flash is present on 99 percent of them. Yes, web surfers can download the Silverlight plug-in quickly and for free, but if they don't already have it, there's a risk they won't bother, in which case they won't be able to see your videos at all.

Video in HTML5

The Web is on the cusp of a big change. HTML5 is just around the corner, and it introduces a new, simpler way to incorporate audio and video into your pages. Instead of worrying about the `<embed>` element and browser plug-ins, you can use the properly supported `<video>` element, which looks like this:

```
<video src="ouch.mpg" controls></video>
```

In this example, the word *controls* is important—it tells the browser to add the playback controls that let visitors play the video. If you leave this out, you need to write some JavaScript code to do the same thing.

HTML5 support isn't here yet, at least not for everyone. Internet Explorer users won't get any HTML5 features until IE 9, which hasn't been released at the time of this writing (let alone widely adopted). To deal with this situation, you need to include fallback content nestled inside the `<video>` element. That way, if a browser doesn't understand the `<video>` element, it displays alternate content, such as a link to download the video:

```
<video src="ouch.mpg" controls>
  Click here to download the <a href="ouch.mpg">ouch.mpg</a> video.
</video>
```

Of course, a link isn't nearly as good as having a real video window. To compensate, you can put a Flash `<embed>` element inside the `<video>` element (assuming you have a Flash video player and another copy of your video in FLV format). Or you could nestle a YouTube window inside, which gets messy, but ensures that your visitors have some way to see your video.

If the `<video>` element seems like more trouble than it's worth—well, it gets worse. At the time of this writing, the makers of the most popular browsers haven't agreed on what video formats they'll support. So if you want to use HTML5 without leaving any browser behind, you need to create several versions of your video file, and write scandalously messy markup using a technique explained at http://camendesign.com/code/video_for_everybody. (But we warned: This website uses HTML5 tags itself, and Internet Explorer owners can't read it in any version except IE 9.)

With all these headaches, why would anyone bother to use HTML5 video at all? One compelling reason is Apple, the high-tech company that's conquering the world with iPhones and iPads. Apple stubbornly refuses to let Flash run on these devices for reasons that are partly technical and partly political. But Apple *is* committed to supporting HTML5 in its Safari browser. In other words, if you want your videos to be viewable on Apple's current crop of mobile devices, you need HTML5, not Flash.

If you want to avoid these problems altogether, YouTube is a far better solution. Its pages are intelligent enough to determine whether a browser needs HTML5 or Flash, and it sends just the right content. And as a side benefit, it gives all your videos free web space and bandwidth.

Uploading Your Videos to YouTube

Before YouTube hit the scene, video clips hadn't really taken off on the Web. They were all-around inconvenient: slow to download, with playback often jerky and sporadic. But today the landscape has shifted. Web connections are faster and browser plug-ins that support movie playback (like Flash and Silverlight) are more common. Ordinary people own all sorts of digital video gadgets that can shoot short movies, from true video cameras to digital cameras, cellphones, and webcams. Popular clips rocket around the world, going from unknown to Internet sensation in a matter of hours. Family members, adventurers, and wannabe political commentators all regularly use video to keep in touch, show their skills, and dish the dirt.

YouTube (www.youtube.com) is at the forefront of this revolution. Despite being a relative web newcomer (YouTube was created in 2005), it currently ranks as the world's third most-popular site (behind Google and Facebook). And YouTube's range of content is staggering. With a quick search, you can turn up both amateur and professional content, including funny home videos, product reviews and announcements, homemade music videos, clips from movies and television shows, and ordinary people spouting off on just about any topic (a trend called *video blogging*).

If you're still considering options for putting your video online, there are three great reasons to use YouTube:

- **It performs well.** YouTube uses Flash to ensure that virtually all browsers can play back its videos. In addition, its videos support *progressive downloading*, which means you can watch the beginning of a video as your browser downloads the rest of it, rather than waiting for the whole enchilada.
- **It extends the reach of your website.** YouTube is one of the most popular sites on the Web. Videos that get lucky can increase their audience from a few people to millions of eager clip-watchers. By putting your movies on YouTube, you increase the odds that someone will discover it and possibly visit your site afterward. For example, many of the most popular clip-makers capitalize on their YouTube popularity by selling themed merchandise on their sites. Others use AdSense (page 385), which includes special ad boxes that can sit unobtrusively at the bottom of a video window.
- **It supports HTML5.** YouTube is an early adopter of HTML5's video features. However, the site's sharp enough to use HTML5-powered pages only when the requesting browser supports HTML5 (otherwise, it sticks with better-supported Flash videos). This sleight-of-hand ensures that owners of devices that don't support Flash (for example, iPads) still get to see your videos.

One disadvantage with YouTube is that you lose control over video quality. YouTube is notorious for applying a heavy dose of compression to shrink video size, making some clips look terrible. The clips that suffer the most are those already encoded, because YouTube encodes them a second time, obliterating fine detail in the process.

In the following sections, you'll see how to upload your first YouTube video, and even learn how to embed it in a window on one of your own pages.

Signing Up with YouTube

Anyone can browse and view YouTube's full catalog of videos (6 million clips at the time of this writing). But to upload your own movies, you need a YouTube account. Here's how to create one:

1. Go to www.YouTube.com. Click the Sign Up link in the top-right corner of the page.
2. Fill in your account information.

You need to supply the usual particulars, including your email address, password, location, and date of birth. Unlike some websites, which identify you solely by your email address, YouTube requires a *user name*, which is a string of letters and numbers like *JoeTheMovieMaker403*. Given the site's popularity, it may take a few tries to find an available name. To find out if a name is available, click the Check Availability link after you type one in.

Note: If you already have a Google Account, you can use that user name and password to log in to YouTube. Scroll down to the bottom-left of the Sign Up page, and then click the "Sign in with your Google Account" link. You still need to pick a YouTube user name and supply your email address, but you'll be able to log in to all the Google services you use with the same email and password combination.

3. Click Create My Account.

YouTube sends a confirmation message to your email address. When you get it, click the link inside to confirm your account.

Preparing a Video

Now you're ready to post a video. But before you do, it helps to understand a bit more about how YouTube works, and the sort of video files it expects. Here are some essential bits of YouTube wisdom:

- YouTube accepts video in virtually any format. That means you probably won't need to convert your file before you submit it. Some of the popular formats YouTube supports are MPEG2, MPEG4, MOV, DivX, WMV, WebM, AVI, and FLV.
- YouTube automatically re-encodes the videos you upload so that web visitors can download it without teeth-gnashing delays. For this reason, YouTube recommends you upload the original version of your clip. In other words, don't re-encode your video. Doing so only wastes your time and lowers the movie's quality because YouTube re-encodes the clip as it prepares it for playback. However, there's a catch: Depending on the length of the clip you want to upload and the video format you use, your original file could be gargantuan (easily running into hundreds of megabytes). Even though YouTube allows uploads of files up to 2 GB, videos of this size aren't practical for everyone (they take forever to upload, for one thing).

Note: If you have a slow web connection, or if your Internet service provider limits how much data you can transfer in a month, you might need to shrink your video files. In this case, the best option is to try and ratchet the size down at the source; in other words, use the lowest possible recording format. But if your files are still too big, or if you need to record higher-quality files so you can use them for other purposes (like burning DVDs), you may be forced to re-encode your video using the process described on page 503.

- YouTube plays back both standard and widescreen video, at different resolutions. If you shoot a standard-shaped video (that is, in the 4:3 aspect ratio), recording at a resolution of 480×360 works well. For widescreen videos, recording at a resolution of 640×360 is good. YouTube also supports higher resolutions (640×480 and 1280×720). Upload video at these resolutions, and YouTube creates different quality copies so viewers can choose which version to watch based on their connection speed. The important thing is not to convert widescreen video to standard video, or vice versa, because you'll end up with a small video surrounded by black bars to compensate for the different screen shapes.
- Read the YouTube Help section. From time to time, YouTube changes its recommendations or introduces new supported formats. To get the lowdown before you upload, visit <http://tinyurl.com/66onkub>.

Uploading a Video

Once your video's ready, it's time to put it online. The process is refreshingly straightforward:

1. Head back to YouTube and sign in.

You'll see that YouTube offers plenty of features to help you track down the videos you like. It keeps track of the videos you watch, recommends related videos, and lets you subscribe to specific video groups. Right now, ignore these features and concentrate on adding your own video creation to the mix.

2. Click the Upload link at the top of the page.

You'll see an upload "drop box" and a Browse button.

3. Drag and drop a file into the upload box, or click the "Upload video" button to browse for it on your hard drive.

Either way, you can upload more than one video at a time.

As soon as you pick your clips, YouTube begins uploading them (Figure 17-10). Now you can take your time entering the video information, because you started the time-consuming upload process.

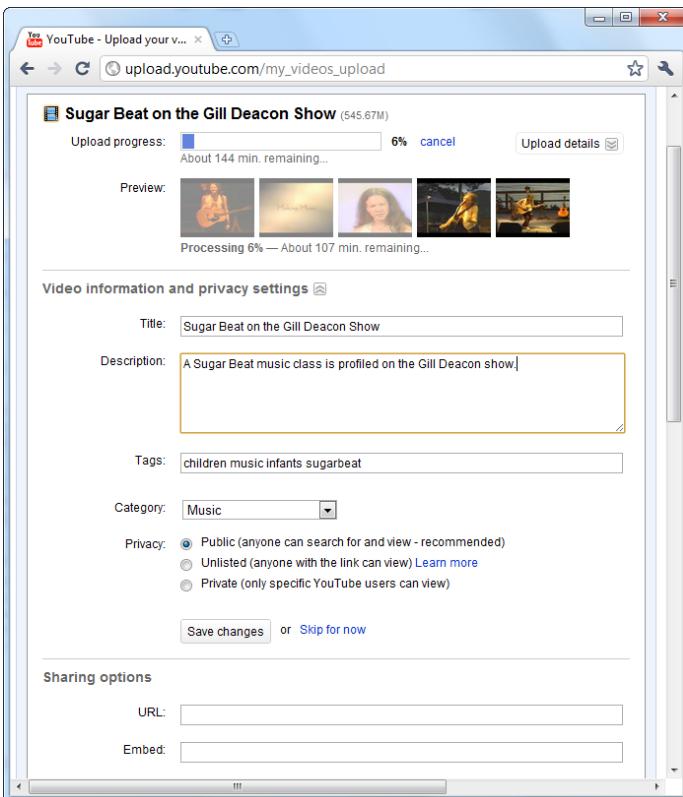


Figure 17-10:

As YouTube uploads this file, it provides an estimate of how long the total transfer will take. Once it uploads a suitable chunk, it starts processing the file. You'll see the occasional thumbnail appear just under the upload progress meter.

4. Fill in the details about your video.

You need to supply a title and description, which YouTube displays on your video page and in its search results. You also need to specify a category for your video, and add one or more *tags*. When other people search YouTube using keywords that match your tags, there's a better chance that your video will turn up in the search results.

Note: Don't worry if the information you enter isn't perfect. You can change it at any time. Just find your video in the My Videos section, select it, and then click Edit.

5. Choose a Privacy setting.

A public video turns up in YouTube search results and anyone can watch it.

Anyone can also watch an unlisted video, too, but they need the right URL to find it—the video won't turn up in an ordinary search.

If you mark a clip as Private, only YouTubers you explicitly identify can see it, and only after they log in to YouTube.

6. Wait.

YouTube says it typically takes 1 to 5 minutes to upload each megabyte of video if you have a high-speed connection, so this is a good time to get a second cup of coffee.

While you're waiting, you can scroll down to the next upload box, pick another video file, and start uploading it as well. Or, if you've given up completely, click Cancel to stop the upload.

When your video finishes uploading and YouTube converts it, you'll see a message at the top of the upload page (see Figure 17-11).

The screenshot shows a web browser window for YouTube's upload interface. At the top, it says "YouTube - Upload your v..." and the URL "upload.youtube.com/my_videos_upload". The main content area displays a video thumbnail for "Sugar Beat on the Gill Deacon Show" (545.67M). Below the thumbnail, a progress bar indicates "Upload progress: 100%". A link "Upload details" is visible. Underneath the thumbnail, five smaller preview images are shown, with the caption "Processing 6% — About 107 min. remaining...". Below the preview section is a form for "Video information and privacy settings". It includes fields for "Title" (Sugar Beat on the Gill Deacon Show), "Description" (A Sugar Beat music class is profiled on the Gill Deacon show.), "Tags" (children music infants sugarbeat), "Category" (Music), and "Privacy" (set to Public). At the bottom of this section are "Save changes" and "Skip for now" buttons. The final section, "Sharing options", contains a "URL" field with the value "http://www.youtube.com/watch?v=qvg2PvXqYI4" and an "Embed" field containing the HTML code <object width="425" height="344"><param name="movie" value="http://www.youtu...". A red box highlights the "Embed" field and its contents.

Figure 17-11:
YouTube gives you two important pieces of information at the end of every upload: a link you can follow to watch your video and a snippet of HTML markup you can paste into any web page to embed your video.

7. Optionally, edit your video to configure a few more settings.

To do this, click your user name (at the top-right of the page) and choose My Videos. Find the video you just uploaded, and click the Edit button next to it.

You can edit the information you supplied when you first uploaded the video, and set the following additional options:

- Pick a **Video Thumbnail**, a single frame from your video that will appear when YouTube lists your video in a search result.
- Use **Date and Map** section's options to identify when and where you recorded your video.
- Use the **Comments**, **Comment Voting**, **Video Responses**, and **Ratings** options to control how people can respond to your video. For example, you can ban people from commenting, or allow only comments you approve. You can also specify whether people can vote on other people's comments, post a video response (a video linked to yours), and rate your videos (using a thumbs up or thumbs down button). Ordinarily, YouTube allows all these options, giving the site a somewhat raucous community atmosphere. In some cases, it makes sense to limit comments and ratings (for example, if you have a sensitive video and you're worried about attracting abusive comments). But the vast majority of videos on YouTube allow comments (and are heavily commented). Videos that don't are likely to be ignored.
- Use the **Embedding** section to control whether other people can embed your video on their web pages. But realize that if you don't allow this, it not only stops *other* people from showcasing your video, it prevents *you* from embedding your own video on your website.

Tip: You can handle other management tasks on the Uploaded Videos page as well. For example, select a video and then click Delete to remove it, Edit to change the video information and options you specified when you uploaded the clip, and Insight to get some fascinating statistics on the people who've seen your video.

8. When you finish, click Save Changes.

Now your video is completely configured and live before the entire web world.

Watching a Video

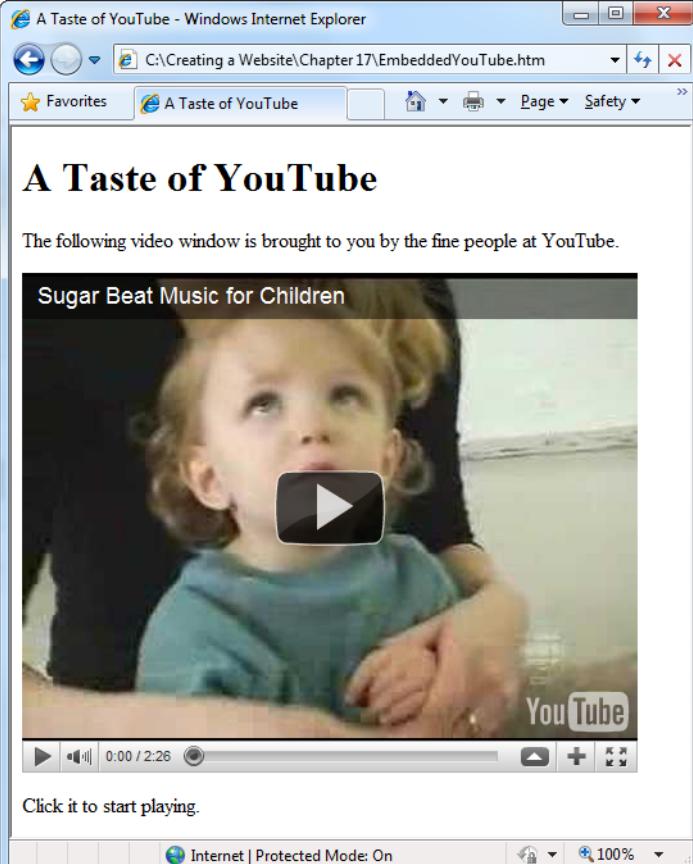
Once your video is ready, you can watch it in several ways:

- You can search for it on YouTube.
- You can browse through the videos in your account. Log into YouTube, click your user name (at the top-right of the page), and then choose My Videos.
- You can play it back in a YouTube window on one of your web pages. To do so,

copy the markup shown in Figure 17-11 into your page. (If you neglected to copy this markup when you uploaded your video, go to the YouTube page for your video and click the Embed button to get it again.)

- You can put a link on your web page that leads to your YouTube video page. Just visit the page and copy the URL from the address bar.

Of all these options, embedding your video in a YouTube window on your own web page is the most interesting (see Figure 17-12). It lets you combine the look and feel of a self-hosted video with YouTube's high performance and solid browser support. Best of all, embedding videos is as easy as copying a snippet of HTML markup into your page. You simply put this markup where you want the video window to appear. Often, you'll put it in a <div> element, and use style rules to position it (as described in Chapter 9).



The screenshot shows a Windows Internet Explorer window titled "A Taste of YouTube - Windows Internet Explorer". The address bar displays the local file path "C:\Creating a Website\Chapter 17\EmbeddedYouTube.htm". The main content area features a heading "A Taste of YouTube" followed by the text "The following video window is brought to you by the fine people at YouTube." Below this, there is a video player window for a video titled "Sugar Beat Music for Children". The video frame shows a young child with blonde hair looking upwards. A large play button is centered over the video frame. In the bottom right corner of the video frame, the YouTube logo is visible. Below the video frame is a control bar with icons for play, volume, and other media controls, and a progress bar showing "0:00 / 2:26". At the bottom of the page, there is a call-to-action text "Click it to start playing." The status bar at the bottom of the browser window shows "Internet | Protected Mode: On" and "100%".

Figure 17-12:
Embedding lets you watch your video in a page of your own devising.

Here's the complete markup that creates the page in Figure 17-12:

```
<!DOCTYPE html>

<html>
<head>
    <title>A Taste of YouTube</title>
</head>

<body>
    <h1>A Taste of YouTube</h1>
    <p>The following video window is brought to you by the fine people
        at YouTube.</p>

    <object width="425" height="344">
        <param name="movie"
            value="http://www.youtube.com/v/qxOQIUkRAGQ&hl=en&fs=1"></param>
        <embed src="http://www.youtube.com/v/qxOQIUkRAGQ&hl=en&fs=1"
            type="application/x-shockwave-flash" allowfullscreen="true"
            width="425" height="344">
        </embed>
    </object>

    <p>Click it to start playing.</p>
</body>
</html>
```

Keen eyes will notice that this video window consists of an `<embed>` element wrapped in an `<object>` element. This messy markup is great for browser compatibility, because browsers that support the `<object>` element will ignore the content inside, and browsers that don't recognize the `<object>` element will use the `<embed>` element instead.

Tip: To change the color of the border around your video window, make it start playing automatically, give it support for full-screen mode, or tweak one of several other details, you need to adjust the parameters inside the markup. For the complete scoop, check out <http://tinyurl.com/2kxnv3>.

HTML Quick Reference

HTML is the language of the Web. You can use it to create any web page, whether you're promoting a local bake sale or running a Fortune 500 company. Chapter 2 introduced HTML in detail, and since that point you've steadily added to your arsenal of HTML elements.

This appendix provides a quick reference of all the HTML elements you've seen in this book (and a few more). Each entry features a brief description of what the element does, and many entries provide cross-references to more detailed examples in the book. You'll also get a preview of the most usable new elements in HTML5 (those that can be coaxed into working with Internet Explorer 8 and older), and a quick refresher of HTML character entities, which let you display special characters on a web page.

HTML Elements

As you already know, the essential idea behind the HTML standard is *elements*—specialized codes in angle brackets that tell a browser how to format text, when to insert images, and how to link different documents together. Throughout this book, you examined just about every important HTML element in use today, not including the so-new-they-break-your-browser goodies in HTML5.

The elements in this reference are arranged in alphabetical order. At the beginning of each section are two key details about the element:

- **Block or inline.** You can put a block element directly in the `<body>` section of a web page. You can't do that with an inline element; you need to put it *inside* another block element. Some elements are even more restricted and can be placed only in certain types of elements. For example, the `<area>` element, which defines

an image-map hotspot, is restricted to the `<map>` element that defines an image map (page 226).

Note: In the official HTML5 lingo, block elements are called *flow elements*, and inline elements are called *phrasing elements*.

- **Container or standalone.** A container element can hold text or other elements inside it. A standalone element has to remain empty, so you usually write it as one tag with the empty element syntax (page 28).

This reference doesn't include any of the new elements from HTML5—you'll find those in a separate section starting on page 535.

<a> (Anchor Element)

Inline Element, Container Element

The anchor element (`<a>`) has two roles. The most common is to create a link that, when you click it, takes you from one page to another. To insert this type of link, you supply the destination URL using the `href` attribute and put the clickable link text between the opening and closing tags:

```
<a href="LinkedPage.htm">Click Me</a>
```

When setting the `href` attribute, you can use a *relative* URL, which points to a page on your own website, or an *absolute* URL, which starts with `http://` and can point to any page on the Web. For a review of the differences between relative and absolute links and when to use each, see page 215.

Creating clickable image links is just as easy as creating clickable text links. The trick is to put an `` element inside an `<a>` element, like this:

```
<a href="LinkedPage.htm"></a>
```

Finally, you can create a link that, instead of sending a visitor to a new page, pops up an email message with the address information filled in. You do this by creating a *mailto* link, as shown here:

```
<a href="mailto:me@myplace.com">Email Me</a>
```

For more information about the ins and outs of the mailto link, see page 224.

You can also apply the `target` attribute to an anchor, which instructs a browser to open the destination page in a specific frame or in a new browser window, like this:

```
<a href="LinkedPage.htm" target="_blank">Click Me</a>
```

The anchor element also works with *bookmarks*, letting you lead website guests to a specific spot on a web page. To create a bookmark, you simply need an element with a name set in its `id` attribute, like this:

```
<h2 id="Canaries">Pet Canaries</h2>
```

Once you create a bookmark, you can write a URL that points to that bookmark. The trick is that you need to add the bookmark information to the end of the URL. To do this, add the number-sign symbol (#) followed by the bookmark name, as shown here:

```
Learn about recent developments in <a href="sales.htm#Canaries">canary sales</a>.
```

You can learn more about bookmarks and ordinary links in Chapter 8.

<address>

Block Element, Container Element

Webmasters use the address element only occasionally; it provides a way for website visitors to contact the authors of the web page. This contact information could be an email address or web link (the two most common options), or a postal address. Here's an example:

```
Our website is managed by:  
<address>  
  <a href="mailto:jsolo@mysite.com">John Solo</a>,  
  <a href="mailto:lcheng@mysite.com ">Lisa Cheng<a>, and  
  <a href="mailto:rpavane@mysite.com ">Ryan Pavane</a>.  
</address>
```

However, if you have other address information on your web page, say your store's physical address, don't use the <address> element. By convention, it's reserved for address details for the person or people who maintain the web page.

Most browsers format addresses in italics, just as though you had used the <i> element. The only value in using the <address> element is that it lets automated programs that scan web pages extract useful address information.

<area> (Image Map)

Allowed in <map> Only, Standalone Element

The <area> element defines a clickable region (known as a *hotspot*) inside an image map (which you generate with the <map> element; see page 527). When defining an area, you need to supply the target URL (using the *href* attribute), the type of shape (using the *shape* attribute), and the coordinates of that shape (using the *coords* attribute). For *shape*, you can specify a circle, square, or polygon.

For a circle, specify the coordinates in this order: center point (x-coordinate), center point (y-coordinate), radius. For any other shape, supply the corners in order as a series of x-y coordinates, like this: x1, y1, x2, y2, and so on. Here's an example that creates a square hotspot:

```
<area href="page1.htm" shape="square" coords="5,5,95,195" alt="A clickable square" />
```

The square is invisible. If you click anywhere inside this square, you'll go to *page1.htm*. For more information, see the `<map>` element on page 527. For a full-fledged image map example, see page 226.

** (Bold Text)**

Inline Element, Container Element

The bold text element displays text in boldface. The official rules suggest that you use `` for “stylistically offset” text; text that should be presented in bold but that doesn’t have greater importance than the rest of your content. This could include keywords, product names, and anything else that would be bold in print.

Make sure you buy the `Super-Fraginator` today!

If you want to format text in bold *and* convey additional importance, `` is the recommended choice (although the visual result is the same).

<base> (Base URL)

Allowed in `<head>` Only, Standalone Element

The base element defines a document’s *base URL*, which is a web address used to interpret all relative paths. You have to place the `<base>` element in the `<head>` section of a page, and you can use two attributes—`href` (which supplies the base URL) and `target` (which supplies a target frame for links).

For example, if you have a link that points to a file named *MySuperSunday.htm* and the base URL is *http://www.SundaysForever.com/Current/*, the browser interprets the link as *http://www.SundaysForever.com/Current/MySuperSunday.htm*. Web-heads rarely use the base URL this way because it almost always makes more sense for the browser to use the current page as a starting point for all relative URLs. In other words, if you’re looking at *http://www.SundaysForever.com/Current/Intro.htm*, the browser already knows that the base URL is *http://www.SundaysForever.com/Current/*. For more information about the difference between absolute and relative links, see page 215.

<blockquote> (Block Quotation)

Block Element, Container Element

The `<blockquote>` element identifies a long quotation (which stands on its own, separate from other paragraphs) as a block element:

```
<blockquote><p>It was the best of times, it was the worst of times.</p>
</blockquote>
```

Usually, browsers indent the `<blockquote>` element on the left and right side. However, you shouldn’t use `<blockquote>` for formatting alone. Instead, use it where it

makes sense—to highlight a passage quoted from a book. As with any element, you can use a style sheet rule to change the way the browser formats <blockquote> text.

If you want to put a brief quotation inside a block element (like a paragraph), use the <q> element instead of <blockquote>.

<body> (Document Body)

Allowed in <html> Only, Container Element

The <body> element is a basic part of the structure of any HTML document. You put it immediately after the <head> section ends, and it contains all the content of your web page, including its text, image URLs, tables, and links.

 (Line Break)

Inline Element, Standalone Element

The line break (
) is an inline element that forces the text following it onto a new line, with no extra spacing. For example, you can use the
 element to split address information in a paragraph:

```
<p>Johny The Fever<br />
200 Easy Street<br />
Akimbo, Madagascar</p>
```

<button> (Button)

Inline Element, Container Element

The <button> element lets you create a clickable button within a form, with any content you want inside of it (for example, you can put a phrase or an image between the <button> element's start and end tags). As with any other form control, you need to supply a unique name and a value that the form will submit when a visitor clicks the button. You put the button content between the opening and closing tags:

```
<button name="submit" value="order" type="button">Place Order</button>
```

You can create three types of buttons, depending on the value you choose for the *type* attribute. A value of *button* creates an ordinary button with no built-in smarts (add JavaScript code to make it do something). A *reset* button clears the input controls in a form, and a *submit* button sends the form data back to the web server, which is useful if you create an application that runs on the server.

The <button> element is more powerful than the <input> element for creating buttons, because it puts whatever content you want on the face of the button, including images.

```
<button name="submit" value="order" type="button">
  
</button>
```

<caption> (Table Caption)

Allowed in <table> Only, Container Element

The <caption> element defines the title text for a table. If you use it, you have to make it the first element in a <table> element:

```
<table>
  <caption>Least Popular Vacation Destinations</caption>
  ...
</table>
```

HTML applies no automatic formatting to the caption; it simply positions the caption at the top of a table as ordinary text (and wraps it to fit the width of the table). You can apply whatever formatting you want through style sheet rules.

<cite> (Citation)

Inline Element, Container Element

The <cite> element identifies a *citation*, which is a reference to a book, print article, or other published resource.

```
<p>Charles Dickens wrote <cite>A Tale of Two Cities</cite>. </p>
```

Usually, browsers render the <cite> element as italic text. But you shouldn't use the <cite> element for formatting alone. Instead, use it when it makes sense (for example, when you refer to a published work you quote) and add style sheet rules that apply the specific formatting you want.

<code>

Inline Element, Container Element

Use the <code> element to wrap small snippets of example code (for example, in a web page that presents a programming tutorial). Browsers display this code in a monospaced font.

<dd> (Dictionary Description)

Allowed in <dl> Only, Container Element

The <dd> element identifies the description in a dictionary list. For more information, see the simple example under the <dl> element description below, or refer to page 121.

 (Deleted Text)

Block Element or Inline Element, Container Element

Webmasters rarely use the `` element; it identifies text that was present but has now been removed. Browsers that support this element display crossed-out text to represent the deleted material. Another element web-heads sometimes use to indicate a revision trail is `<ins>` (see page 526).

<dfn> (Defined Term)

Inline Element, Container Element

Site authors rarely use the `<dfn>` element; it indicates the defining instance of a term. For example, the first time you learn about a new term in this book, like *froopy*, it's italicized. That's because it's the defining instance, and a definition usually follows. Browsers render the `<dfn>` element in italics.

<div> (Generic Block Container)

Block Element, Container Element

The division element groups together one or more block elements. For example, you could group together several paragraphs, a paragraph and a heading, and so on. Here's an example:

```
<div>
  <p>...</p>
  <p>...</p>
</div>
```

On its own, the `<div>` element doesn't do anything. However, it's a powerful way to apply style sheet formatting. In the example above, you can apply formatting to the `<div>` element and the browser passes that formatting along to the two nested paragraphs (assuming the style properties you're using support inheritance, as described on page 145).

To learn more about using the `<div>` element to apply style rules, see page 179. You should also refer to the `` element (page 531), which applies formatting inside a block element. And if you want to apply more meaning to your containers, consider replacing your `<div>` with one of the HTML5 semantic elements discussed starting on page 535.

<dl> (Dictionary List)

Block Element, Container Element

The `<dl>` element defines a definition list (also known as a dictionary list), which is a series of terms, each followed by a definition in an indented block of text that appears immediately below it. In theory, you could put any type of content in a dictionary list, but it's recommended that you follow its intended use and include a list of points and explanations. Here's an example:

```
<dl>
  <dt>tasseomancy</dt>
  <dd>Divination by reading tea leaves.</dd>
  <dt>tyromancy</dt>
  <dd>Divination by studying how cheese curds form during cheese making.</dd>
</dl>
```

<dt> (Dictionary Term)

Allowed in <dl> Only, Container Element

The <dt> element identifies the term in a dictionary list. For more information, see the simple example under the <dl> element description above, or refer to page 121.

** (Emphasis)**

Inline Element, Container Element

The element has the same effect as the <i> (italic text) element, but a slightly different meaning. It's for emphasized text that would have a different inflection if read out loud, like this:

Make sure you don't use the wrong element for your italics.

By comparison, the <i> element is for italicized text that doesn't have this emphasis.

Using style sheet rules you can change the formatting of the element, and emphasize its content in a way that doesn't use italic formatting (like by coloring the text red).

<embed> (Embedded Object)

Inline Element, Container Element

The <embed> element embeds special objects in your page, like audio, video, and even *applets* (miniature programs that run inside a web page). For example, you might use an <embed> element to put a background music player or YouTube video in your web page, as described in Chapter 17.

The <embed> is closely related to the <object> element, although the syntax is different. Because some very old browsers support <embed> but not object, they are sometimes used in conjunction, so that the <embed> element provides fallback content if the <object> element doesn't work, as shown on page 514.

<form> (Interactive Form)

Block Element, Container Element

The <form> element creates an interactive form, where you put graphical widgets like text boxes, checkboxes, selectable lists, and so on (represented by the <input>, <textarea>, <button>, and <select> elements, respectively). By putting these widgets

in a `<form>` element, you can create pages that collect information from visitors and submit this information to a web application. Web applications are outside the scope of this book, but you can see how to use a `<form>` element with JavaScript on page 456, and you can consider free form submission services on page 326.

<h1>, <h2>, <h3>, <h4>, <h5>, <h6> (Headings)

Block Element, Container Element

Headings are section titles. They use bold lettering at various sizes. The size of the heading depends on the heading level. The six heading levels start at `<h1>` (the biggest) and move down to `<h6>`. Both `<h5>` and `<h6>` are actually smaller than regularly sized text. Here's an `<h1>` element in action:

```
<h1>Important Information</h1>
```

When you use headings, always make sure your page follows a logical structure that starts with `<h1>` and gradually works its way down to lower heading levels. Don't start with `<h3>` just because the formatting looks nicer. Instead, use style sheets to change the formatting of each heading to suit you, and use the heading levels to delineate the structure of your document.

<head> (Document Head)

Allowed in `<html>` Only, Container Element

The `<head>` section goes before the `<body>` section of a page. While the `<body>` element contains the web page content, the `<head>` element includes other information, like the web page title (the `<title>` element), descriptive metadata (one or more `<meta>` elements), and styles (the `<style>` or `<link>` elements).

<hr> (Horizontal Rule)

Block Element, Standalone Element

The `<hr>` element defines a horizontal rule (a solid line) that you use to separate block elements:

```
<p>...</p>
<hr />
<p>...</p>
```

Although the `<hr>` element still works perfectly well, HTML whizzes prefer using border settings in a style sheet rule to get much more control over the line style and its color. Here's an example that defines a style sheet rule for a solid blue line:

```
.border { border-top: solid medium navy }
```

And here's how you could apply it:

```
<p>...</p>
<div class="border"></div>
<p>...</p>
```

For more information about the style sheet border settings, refer to page 172.

<html> (Document)

The <html> element is the first element in any HTML document. It wraps the rest of the document. If you create an ordinary web page, the <html> element contains two other essential ingredients—the <head> element, which defines the title, metadata, and linked style sheets; and the <body> element, which contains the actual content.

<i> (Italic Text)

Inline Element, Container Element

The <i> element displays text in italics, without conveying any extra emphasis or inflection. Here's an example:

```
The mattress label says <i>do not remove under penalty of law</i>
```

If you want to provide italics that aren't just typographic, but also suggest emphasis, the element is a better choice. However, both elements provide the same formatting.

<iframe> (Inline Frame)

Inline Element, Container Element

The <iframe> element creates an *inline frame*—an embedded, scrollable window that displays a web page inside another one. You supply the attributes *src* (the page you want your browser to display in the frame), *name* (the unique name of the frame), and *width* and *height* (the dimensions of the frame in pixels). You can also turn off the border by setting the *frameborder="0"* attribute, or turn off scrolling by adding the *scrolling="no"* attribute. Here's one use of the <iframe> element:

```
<iframe src="MyPage.html" width="100" height="250"></iframe>
```

You can put content inside the <iframe> element that a browser will display if it doesn't support the <iframe> element:

```
<iframe src="MyPage.html" width="100" height="250">
  <p>To see more details, check out <a href="MyPage.html">this page</a>.</p>
</iframe>
```

 (Image)

Inline Element, Standalone Element

The element points to a picture file you want to display in a page. The *src* attribute identifies the picture using a relative or absolute link (see page 215). The *alt* attribute supplies text that a browser displays if it can't display the picture.

```

```

Internet Explorer displays alternate text in a pop-up text box, while some more standards-aware browsers (namely Firefox) don't. In either case, you can supply a pop-up text box in just about any browser using the *title* attribute. This is the best way to add pop-up text to an image.

The `` element also supports *height* and *width* attributes you can use to explicitly size a picture:

```

```

In this example, the picture has a width of 100 pixels and a height of 150 pixels. If these dimensions don't match the actual size of the source picture, your browser stretches and otherwise mangles the picture to match the dimensions.

Never use the *width* and *height* attributes to resize an image; instead, make those kinds of edits in a proper image-editing program. You can use the *width* and *height* attributes to tell a browser how big your picture is so it can lay out the page before it downloads the whole image, and so your layout is preserved even if the browser can't find your picture file.

To learn more about supported image types, how to organize pictures on a page, and where to find the best material, refer to Chapter 7.

Finally, you can create clickable regions on an image by defining an image map, and then link that image map to your image with the *usemap* attribute of the `` element. For more information, see the `<map>` section (page 527).

<input> (Input Control)

Allowed in `<form>` Only, Standalone Element

The `input` element is the most common ingredient in an HTML form (which is itself represented by the `<form>` element). The `<input>` element can represent different onscreen widgets (called *controls*) that collect information from a web visitor.

The *type* attribute specifies the kind of control you want to create. Table A-1 lists the most common types. Additionally, you should give every control a unique name using the *name* attribute.

Table A-1. HTML form controls.

Control	HTML Element	Description
Single-line text box	<code><input type="text" /></code>	Shows a text box where a visitor can type in any text.
Password text box	<code><input type="password" /></code>	Shows a text box where a visitor can type in any text, but the browser doesn't display the text. Instead, it displays an asterisk (*) in place of every letter, to hide the text from prying eyes.
Checkbox	<code><input type="checkbox" /></code>	Shows a checkbox you can set as turned on or off.

Control	HTML Element	Description
Radio button	<input type="radio" />	Shows a radio button (a circle you can set as turned on or off). Usually, you have a group of radio buttons next to each other, in which case a visitor selects exactly one.
Submit button	<input type="submit" />	Shows a standard push button that submits a form and all its data.
Reset button	<input type="reset" />	Shows a standard push button that clears visitor selections and entered text in all the input controls of the form.
Image button	<input type="image" />	Shows a submit button with a difference—you supply its visuals as a picture. To specify the picture file you want, set the <code>src</code> attribute.
Ordinary button	<input type="button" />	Shows a standard push button that doesn't do anything unless you hook it up to some JavaScript code (Chapter 15).

Here's an `<input>` element that creates a text box. When a visitor submits the page, whatever they typed into the box will be sent, along with the descriptive identifier `LastName`:

```
<input type="text" name="LastName" />
```

Of course, forms are only useful if you have code that processes their data. On page 456, you saw how site guests can interact with forms using JavaScript, but forms are more commonly used with server-side web applications. For example, a server-side script might receive some visitor information from a form and store it in a database

<ins> (Inserted Text)

Block Element or Inline Element, Container Element

The `<ins>` element identifies newly inserted text (for example, you can use it to show how a web page has changed in its most recent edit). By default, the text inside the `<ins>` element is underlined. Because of its specialized purpose, you'll rarely come across pages that use the `<ins>` element.

You can use the `<ins>` element around block elements or inside a block element. The `` element is another revision element, and you might want to use it in conjunction with `<ins>`.

** (List Item)**

Allowed in `` and `` Only, Container Element

The `` element represents a single item in an ordered (numbered) or unordered (bulleted) list. For more information, see the `` element for ordered lists (page 528) and the `` element for unordered lists (page 534).

<link> (Document Relationship)

Allowed in <head> Only, Standalone Element

The <link> element describes a relationship between the current document and another document. For example, you might use it to point to the previous version of the current document. More commonly, you use it to point to an *external style sheet* that provides formatting instructions for the current page. You always put the <link> element in the <head> section of a page. Here's one possible use:

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

By using external style sheets, you can define your styles in one file and use them on multiple pages. Chapter 6 has much more on style sheets and how to use them.

<map> (Image Map)

Inline Element, Container Element

The <map> element defines an *image map*—a picture with one or more clickable regions. When you create an image map, you assign a unique name that identifies the map using the *name* attribute. You then add one <area> element inside the <map> element for each clickable region, specifying the coordinates of the clickable area and the destination URL. (See the <area> element on page 517 for more on how the *coords* attribute works.) Here's an example of an image map with three clickable regions:

```
<map id="Three Squares" name="ThreeSquares">
  <area href="page1.htm" shape="square" coords="5,5,95,195"
    alt="Square #1" />
  <area href="page2.htm" shape="square" coords="105,5,195,195"
    alt="Square #2" />
  <area href="page3.htm" shape="square" coords="205,5,295,195"
    alt="Square #3" />
</map>
```

Finally, to use your image map, you need to apply it to an image using the *usemap* attribute. The *usemap* attribute matches the name of the map, but starts with a number sign (#), which tells browsers that the image map is on the current page:

```

```

You can't see the clickable regions of an image map (unless you outline them in the image). However, when you hover over a hotspot, your mouse pointer changes to a hand. Clicking a hotspot has the same effect as clicking an ordinary <a> link—you immediately go to the new URL. For a full-fledged image map example, see page 226.

<meta> (Metadata)

Allowed in <head> Only, Standalone Element

Meta elements let you embed descriptive information in your web pages. Your visitors never see this information, but automated programs like web search engines can

find it as they scan your site. You add metadata by placing `<meta>` elements in the `<head>` section of your page.

Every `<meta>` element includes a *name* attribute (which identifies the type of information you're adding) and a *content* attribute (which supplies the information itself). Although you can have an unlimited number of potential `<meta>` elements, the two most common are description and keywords, because some search engines use them:

```
<meta name="description" content="Sugar Beat Music for Children offers age-appropriate music classes for children 4 months to 5 years old" />
```

Page 287 describes meta elements in more detail, and explains how search engines use them.

<noscript> (Alternate Script Content)

Block Element, Container Element

The `<noscript>` element defines the content a browser should display if it can't run a script. The `<noscript>` element should immediately follow the `<script>` element. The most common reason a browser can't run a script is because a web visitor has specifically turned off the script feature through browser settings.

For more information about scripts, refer to Chapter 15.

<object> (Embedded Object)

Inline Element, Container Element

The `<object>` element embeds specialized objects in your page, like audio, video, and even *applets* (miniature programs that run inside a web page). For example, you might use an `<object>` element to place a Flash movie inside a web page, as described in Chapter 17.

** (Ordered List)**

Block Element, Container Element

An ordered list starts with the `` element and contains multiple list items, each of which you represent with an `` element. In an ordered list, your browser numbers each item in a list consecutively, using your choice of numbers, letters, or roman numerals.

Here's a simple ordered list that numbers items from 1 to 3:

```
<ol>
  <li>Buy bread</li>
  <li>Soak stamps off letters</li>
  <li>Defraud government with offshore investment scheme</li>
</ol>
```

To start at a number other than 1, use the *start* attribute and supply the starting number. To change the list's numbers or letters format, use the *type* attribute with one of these values: *1* (numbers), *a* (lowercase letters), *A* (uppercase letters), *i* (lowercase roman numerals), *I* (uppercase roman numerals).

HTML5 adds a feature for backward-counting lists. To create one, you add the *reversed="reversed"* attribute to the `` element. However, no browser currently supports this trick.

For more information about ordered lists, see page 119.

<option> (Menu Option)

Allowed in `<select>` Only, Container Element

The `<option>` element defines an item in a selectable list control, inside a `<select>` element. For example, if you want to create a drop-down menu that lets visitors choose a color from a list of options including the entries Blue, Red, and Green, you need one `<select>` element with three `<option>` elements inside it.

When you define the `<option>` element, you can use the *selected="selected"* attribute to tell a browser to select this item when it shows the page for the first time. You can also use the *value* attribute to associate a unique identifying piece of information with an option, which is included with the form data when a visitor submits the form.

For a basic example, see the description of the `<select>` element.

<p> (Paragraph)

Block Element, Container Element

The paragraph element contains a paragraph of text:

```
<p>It was the best of times, it was the worst of times ...</p>
```

Because paragraphs are block elements, a browser automatically adds a line break and a little extra space between two paragraphs, or between a paragraph and another block element, like a list or heading.

Browsers ignore empty paragraphs. To create a blank paragraph that takes up the normal amount of space, use a nonbreaking space like this:

```
<p>&nbsp;</p>
```

<param> (Object Parameter)

Allowed in `<object>` Only, Standalone Element

The `<param>` element defines extra information in an `<object>` element, which a browser sends to an applet or plug-in.

<pre> (Preformatted Text)

Block Element, Container Element

Preformatted text breaks the normal rules of HTML formatting. Inside a `<pre>` element, a browser duplicates every space and line break it sees, retaining the formatting you used originally. Additionally, the browser puts all the content in a monospaced font (typically Courier), which means the results aren't always pretty. The `<pre>` element is an easy and quick way to get text to appear exactly the way you want it, which is useful if you want to represent visual poetry or display a snippet of programming code. However, you shouldn't use it to align large sections of ordinary text; use CSS positioning rules (see Chapter 9) for those tasks.

```

<pre>
    Tumbling-hair
        picker of buttercups
            violets
    dandelions
    And the big bullying daisies
        through the field wonderful
    with eyes a little sorry
    Another comes
        also picking flowers
</pre>

```

<q> (Short Quotation)

Inline Element, Container Element

The `<q>` element defines a short quotation inside another block element, like a paragraph.

`<p>As Charles Dickens once wrote, <q>It was the best of times, it was the worst of times</q>.</p>`

Usually, browsers render the `<q>` element as italic text and some browsers, like Firefox, add quotation marks around the text inside. However, don't use the `<q>` element for formatting alone. Instead, use it to identify quotations in your text, and then add style sheet rules to apply the formatting you want.

If you want a longer quotation that stands on its own as a block element, use the `<blockquote>` element instead (page 518).

<samp> (Sample Output)

Inline Element, Container Element

You can use the `<samp>` element to mark up computer output—for example, text that's displayed in a console window after you run some sort of management script. This rarely used element simply formats its contents with a monospaced font, like `<pre>` and `<code>`.

<script> (Client-Side Script)

Block Element, Container Element

The <script> element includes a client-side script inside your web page. A script is a set of instructions written in a simplified programming language like JavaScript. Web designers use scripts to create more interactive web pages by adding effects like buttons that change color when you mouse over them. To learn some of the basics of JavaScript and see scripts in action, check out Chapter 15.

<select> (Selectable List)

Allowed in <form> Only, Container Element

The <select> element defines a list control inside a form. Your visitor can select a single item from the list (or multiple items, if you add the *multiple="multiple"* attribute). You use the *name* attribute to uniquely identify this control, as in the following example:

```
<select name="PromoSource">
  <option value="Ad">Google Ad</option>
  <option value="Search">Google Search</option>
  <option value="Psychic">Uncanny Psychic Intuition</option>
  <option value="Luck">Bad Luck</option>
</select>
```

Ordinarily, controls create selection lists as drop-down menus. However, you can create a scrollable list box using the *size* attribute. Just specify the number of rows you want to show at once:

```
<select name="PromoSource" size="3">
  ...
</select>
```

For an example of a form, refer to page 456.

<small> (Small Print)

Inline Element, Container Element

Use the <small> element to hold “small print,” like legalese at the bottom of a contract. Visually, small print steps the text size down one notch, although you can change this effect through a style sheet if it’s not appropriate.

 (Generic Inline Container)

Inline Element, Container Element

Use the element to identify text you want to format inside a block element. For instance, you could format a single word in a paragraph, a whole sentence, and so on. Here’s an example:

<p>In this paragraph, some of the text is wrapped in a span element. That gives you the ability to format it in some fancy way later on.</p>

On its own, the element doesn't do anything. However, it's a powerful way to apply style sheet formatting in a flexible, reusable way.

You should also refer to the <div> element, which can apply formatting to several block elements at once (see page 521).

** (Strong Importance)**

Inline Element, Container Element

The element has the same effect as the (bold text) element, but the official rules of HTML suggest you use it for text that has greater importance than the surrounding words. Here's an example:

I'm very sorry for all the trouble.

If you want bold text that doesn't give text extra emphasis than the surrounding words, you're better off using the element. It also makes perfect sense to change the formatting of the element with a style sheet if you want to emphasize important text differently.

<style> (Internal Style Sheet)

Allowed in <head> Only, Container Element

Use the <style> element to supply CSS (Cascading Style Sheet) rules that format a web page. Always put the <style> element inside the <head> section of a web page.

The <style> element lets you define a style right inside a web page. This is known as an *internal style sheet*. Here's an example that gives <h1> headings fuchsia text:

```
<style>
  h1 { color: fuchsia }
</style>
```

More commonly, you'll use the <link> element instead of the <style> element, so that you can link to a separate file that defines your styles. That way, you can apply the same styles to multiple pages without cluttering up your markup. Chapter 6 has much more about style sheets and how to use them.

<sub> (Subscript)

Inline Element, Container Element

The <sub> element formats text so that it appears smaller and lower (the middle of the text lines up with the bottom of the current line). It's best not to rely on this trick for formatting (use style sheets instead), but it's a handy way to deal with scientific terms like H₂O. Here's how you use it:

Water is H₂O

<sup> (Superscript)

Inline Element, Container Element

The `<sup>` element formats text so that it appears smaller and higher (the middle of the text lines up with the top of the current line). It's best not to rely on this trick for formatting (use style sheets instead), but it's a handy way to deal with exponents like 3³. Here's the `<sup>` element in action:

3³ is 27

<table> (Table)

Block Element, Container Element

The `<table>` element is the outermost element that defines a table. Inside the `<table>` element, you define rows with the `<tr>` element, and inside each row, you use the `<td>` element to define individual cells and specify the content they hold. Here's a basic table:

```
<table>
<tr>
  <td>Row 1, Column 1</td>
  <td>Row 1, Column 2</td>
</tr>
<tr>
  <td>Row 2, Column 1</td>
  <td>Row 2, Column 2</td>
</tr>
</table>
```

It looks like this:

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

See page 124 for more information about creating tables and page 261 for information about sizing them.

<td> (Table Data Cell)

Allowed in `<tr>` Only, Container Element

The `<td>` element represents an individual cell inside a table row (a `<tr>` element). Each time you add a `<td>` element, you create a column. However, it's perfectly valid to have different numbers of columns in subsequent rows (although it might look a little wacky). For a basic table example, see the `<table>` element definition above, and for a detailed table explanation, check out Chapter 9.

<textarea> (Multiline Text Input)

Allowed in <form> Only, Container Element

The <textarea> element displays a large text box in a <form> that can fit multiple lines of text. As with all input controls, you need to identify the control by giving it a unique name. Additionally, you can set the size of the text box using the *rows* and *cols* attributes.

If you want text to appear in the <textarea> element initially, put it in between the start and end tags, like so:

```
<textarea name="Comments">Enter your comments here.</textarea>
```

<th> (Table Header Cell)

Allowed in <tr> Only, Container Element

The <th> element represents an individual cell with table heading text. Use the <th> element in the same way you use the <td> element. The difference is that you usually reserve the <th> element for the first row of a table (because it represents column headings), and <th> text appears boldfaced and centered (which you can tailor using style sheets).

<title> (Document Title)

Allowed in <head> Only, Container Element

The <title> element specifies the title of a web page. The browser displays this text in its title bar and uses it as the bookmark text if a visitor bookmarks the page. You have to put the <title> element in the <head> section of a page.

```
<title>Truly Honest Car Mechanics</title>
```

<tr> (Table Row)

Only Allowed in <table>, Container Element

The <tr> element represents an individual row inside a table (a <table> element). To add cells of information, you need to add the <td> element inside the <tr> element. For a basic table example, see the <table> element definition above.

 (Unordered List)

Block Element, Container Element

An unordered list starts with the element, and includes multiple list items, each of which you represent with a element. The browser indents each item in the list, and draws a bullet next to it.

Here's a simple unordered list:

```
<ul>
  <li>Buy bread</li>
  <li>Soak stamps off letters</li>
  <li>Defraud government with offshore investment scheme</li>
</ul>
```

Webmasters often use the `` to create a menu of commands. (You can also use multiple levels of nested lists to create a hierarchical menu). When using this technique, use a style sheet rule to remove the bullets (set `list-style-type` to `none`) and tweak the `margin` and `padding` settings.

HTML5 Semantic Elements

As you learned in Chapter 2, there's a new standard shaking up the Internet world: HTML5. And although HTML5 is full of improvements, it also comes with a significant caveat. Because it's so new, many browsers don't support it completely. Internet Explorer is a particular laggard; you'll get no support unless you use the brand-spanking-new IE 9.

Because of these browser support issues, web designers need to approach HTML5 with caution. Workaround techniques do allow pages to use many of HTML5's features without leaving older browsers in the dark, however, many of these solutions are more trouble than they're worth. (For example, page 506 talks about the considerable hurdles you need to leap to use HTML5's video support.)

You can use one group of elements right now, however, and without much extra effort. These are HTML5's *semantic elements*, which add meaning to the structure of your pages. Semantic elements let you identify the logical purpose of different portions of your page. For example, you can indicate in your markup where your header is, where you placed your navigation links, and so on. Your visitors never see this information, but you can employ it in a variety of other useful ways. For example, search engines can use it to learn more about your website, screen reading programs can present your content more effectively to people with disabilities, and other tools can extract your data and reuse it in dozens of ways (for example, in a news feed; page 353). All of these scenarios are still evolving, but if you plan to stick with your website for a long time, it's worth getting used to some of these new conventions, so you can benefit from them when HTML5 becomes the official standard.

If you decide to use HTML5's semantic elements, you still need to worry about one issue. Older browsers (like IE 8) won't recognize these elements, so they'll just ignore them. This is fine up to a point, because the semantic elements aren't about formatting; they're about structure. However, many of the new semantic elements are block elements, which means a browser should render them on a separate line on the resulting web page, with a little bit of space between them and the preceding (and following) element. Browsers that don't recognize HTML5 elements won't know to display some of them as block elements, so it's up to you to add a style sheet rule that tells these browsers what to do. Here's a super-rule that applies block display formatting to the nine HTML5 elements in one step:

```
article, aside, figure, figcaption, footer, header, hgroup, nav, section,  
summary {  
    display: block;  
}
```

This style sheet rule won't have any effect on browsers that already recognize HTML5, because they will have already set the *display* property to *block*.

The super style sheet rule above solves part of the problem, but not all of it. It works with old browsers that don't understand HTML5, except Internet Explorer. That's because IE refuses to apply style sheet formatting to elements it doesn't recognize. Fortunately, you have a workaround: You can trick Internet Explorer into recognizing a foreign element by registering it with a JavaScript command. Here's a script block that gives IE the ability to recognize and style the <header> element:

```
<script>  
    document.createElement('header')  
</script>
```

Rather than write this sort of code yourself, you can make use of a ready-made script that does it for you (described at <http://tinyurl.com/nlcjxm>). To use this script, you simply add a reference to it in the <head> section of your page, like this:

```
<head>  
    <title>...</title>  
    <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>  
</head>
```

This grabs the script from the html5shim.googlecode.com web server and runs it before the browser starts processing the rest of the page. The script is short and to the point—it uses the JavaScript trick described above to create all the new HTML5 elements. This lets you format the elements with style sheet rules. Add the super-rule shown above to your style sheet, and the new elements will display as proper block elements. The only remaining task is for you to use the elements, and add your own style sheet rules to format them.

The following sections list all of HTML5's new semantic elements. Notably missing are the HTML5 elements that don't have handy workarounds for old browsers. This includes HTML5's audio and video features (which are still enmeshed in a video codec war, as described on page 506), the new HTML5 widgets that you can use when constructing forms (page 453), and the new <canvas> element that you can use to draw shapes with JavaScript.

Tip: To see the latest version of the HTML5 spec, complete with all the features that leave older browsers out in the cold, see <http://dev.w3.org/html5/markup>. To learn about all of HTML5's new features, including the supporting standards for building better web applications with JavaScript, you can read *HTML5: The Missing Manual*, due out in summer 2011.

<article> (Article)

Block Element, Container Element

Represents whatever you think of as an article—a section of self-contained content like a newspaper article, forum post, or blog entry (not including frills like comments or an author bio).

As with almost all the HTML5 semantic elements, the `<article>` element doesn't apply any built-in formatting. Think of it as a more meaningful version of the generic `<div>` container.

GEM IN THE ROUGH

Structuring a Document with Sectioning Roots

The `<article>` element is an example of a slightly abstract HTML5 concept called *sectioning roots*. Essentially, sectioning roots shape the *outline* of your page. If a page has a good outline, it's easier for other tools and programs to understand its structure.

HTML5 considers the following elements to be sectioning roots: `<body>`, `<section>`, `<article>`, `<aside>`, `<footer>`, `<header>`, and `<nav>`. Each of these elements is the top-level of the page outline. In other words, when you add a sectioning root like `<article>` to your page, you are creating a new outline. (The rest of the outline is created from the headings inside the section. You start with the `<h1>` heading level, then use additional heading levels, like `<h2>`, `<h3>`, and so on, as required.)

The concept of sectioning roots becomes important when you have multiple pieces of self-contained content on the

same page. For example, consider a news site that shows several articles side by side or a blog page that shows several postings at once. In traditional HTML, all this content merges into one poorly structured outline. But with sectioning roots, the HTML markup clearly shows the separation, because each article or posting becomes its own sectioning root.

Unfortunately, the concepts of sectioning roots and outlines are still a bit theoretical, because web page editors, browsers, search engines, and assistive devices (like screen readers), don't yet take advantage of all this information. When they do, sectioning roots will become a key concept in helping web designers create well-structured pages. In the meantime, you can study the outline of any HTML5 page using the outlining tool at <http://gsnedders.html5.org/outliner>.

`<aside>` (Sidebar)

Block Element, Container Element

Represents a complete chunk of content that's separate from the main content of the page. For example, it makes sense to use `<aside>` to create a sidebar with related content or links next to a main article. You can also use it for a block of ads.

`<figcaption>` (Figure Caption)

Block Element, Container Element

The `<figcaption>` element wraps the caption text that goes with a `<figure>` (see below). The goal is to clearly indicate the association between an image and its caption. Of course, `<figcaption>` isn't limited to text alone. You can use any HTML elements that make sense; good choices include links and tiny icons.

<figure> (Figure)

Block Element, Container Element

The <figure> element wraps a picture and its associated caption. Here's an example:

```
<figure>
  
  <figcaption>The bark of a plane tree</figcaption>
</figure>
```

In most cases, you want to use a style sheet class to position the <figure> element. For example, you might choose to float it on the left or right side of your page.

<footer> (Footer)

Block Element, Container Element

The <footer> is the chunk of content that sits at the bottom of a page. It may include small print, a copyright notice, and a small set of links (for example, About Us or Get Support).

<header> (Header)

Block Element, Container Element

The <header> element represents a heading that includes a title and some content. For example, a heading at the beginning of an article might include a title and a by-line, or a title and some links to subtopics in the article. Here's an example:

```
<header>
  <h1>An HTML5 Investigation</h1>
  <p>Prepared by Steven Smith</p>
</header>
```

The <header> can also wrap the heading section of a website (for example, a banner with a company logo). It's perfectly acceptable for a web page to have more than one <header> section, as long as each header belongs to different content.

If you need to combine a title and a subtitle, you need to use the <hgroup> element, because <hgroup> allows multiple heading levels. However, you can also combine <header> and <hgroup>, when it makes sense:

```
<header>
  <hgroup>
    <h1>An HTML5 Investigation</h1>
    <h2>Part 1: Getting the Real Story</h2>
  </hgroup>
  <p>Prepared by Steven Smith</p>
</header>
```

<hgroup> (Header Group)

Block Element, Container Element

The `<hgroup>` element is a specialized form of header that groups two or more heading elements, and nothing else. Its primary purpose is to make a title and a subtitle stand together, like this:

```
<hgroup>
  <h1>An HTML5 Investigation</h1>
  <h2>Part 1: Getting the Real Story</h2>
</hgroup>
```

Structurally speaking, browsers treat the headings grouped together in an `<hgroup>` element as a single super-header. If, however, you were to use the same markup without the `<hgroup>` element, the `<h2>` element is considered to be a subsection under the `<h1>` element. This doesn't affect the way your web page appears in a browser, but it has an effect on the outline of your web page, which can affect other programs that analyze the structure of your page, like web design tools, search engines, and screen readers.

<mark> (Highlighted Text)

Inline Element, Container Element

The `<mark>` element represents a section of text you want to highlight for reference. For example, you could use it to flag changes, mistakes, or keywords.

If you decide to use the `<mark>` element now, you'll need to supply the formatting for browsers that don't already support HTML5. Here's the sort of style rule you need:

```
mark {
  background-color: yellow;
  color: black;
}
```

<nav> (Navigation Links)

Block Element, Container Element

The `<nav>` element represents a section of a page that contains links. These links may point to topics on the current page or to other pages on a website. In fact, it's not unusual to have a page with multiple `<nav>` sections—one for the sitewide navigation menu, another for the topics in the current article, and so on.

It's worth noting that you don't have to wrap every block of links in a `<nav>` element. Instead, web designers generally reserve it for the largest and most important navigational sections on a page. You can place the `<nav>` inside another HTML5 element, as well. For example, if you have a set of navigation links right in the header, you're free to place the `<nav>` element inside the `<header>` element. Or, if you decide the links aren't clearly part of the header, you may choose to add the `<header>` element followed by the `<nav>` element. The HTML5 specification is explicit in saying that decisions like these are a matter of taste.

<section> (Section)

Block Element, Container Element

The <section> element is the most generic type of sectioning root (page 537). It represents some sort of document that should start with a heading of any level.

If possible, you should use a more specific container than <section>. For example, both <article> or <aside> have more specific meanings, which makes them better choices than <section>. If you can't really describe your content as an article or an aside (for example, say it's a patient record pulled out of a database), however, a <section> element is a perfectly reasonable way to identify it.

One confusing detail is that you can use a <section> for *parts* of a page. For example, on your website's home page you could put blog postings, news, and a sidebar of ads into their own separate <section> elements.

Note: As web designers become more accustomed to using elements like <section>, people will begin to settle on a few widely accepted usage patterns. Until then, HTML5 leaves some ambiguity about the best ways to use its new elements.

<time> (Date or Time)

Inline Element, Container Element

Date and time information appears frequently in web pages. For example, it turns up at the end of most blog postings. Unfortunately, there's no standardized way to tag dates, so there's no easy way for other programs (like search engine crawlers) to extract it without guessing. The <time> element solves this problem in two ways. Not only does it allow you to flag an existing date, it also adds a *datetime* attribute that you can use to provide the date in a standardized form that any program can understand.

Here are a few examples of the <time> element at work:

```
<time datetime="2011-11-30">30<sup>th</sup> of November</time>
<time datetime="20:00">8:00pm</time>
<time datetime="2011-11-30T20:30">8:30 PM on November 11, 2011</time>
```

Remember, the information in the *datetime* attribute won't appear on your web page. It's there for other programs to read.

Because the <time> element is purely informational and doesn't have any associated formatting, you can use it with any browser.

HTML Character Entities

HTML character entities are codes that a browser translates into other characters when the browser displays the page. All HTML character entities start with the ampersand (&) and end with the semicolon (;).

There are two principal reasons to use HTML character entities. First of all, you might want to use a character that has a special meaning in the HTML standard. For example, if you type < in an HTML document, a browser assumes you're starting to define an element, which makes it difficult to write a pithy bit of logic like "2 < 3." To get around this, you replace the < symbol with a character entity that *represents* the less-than symbol. The browser then inserts the actual < character you want when it displays the page.

The other reason to use HTML character entities is because you want to use a special character that's not easy to type, like an accented letter or a currency symbol. In fact, characters like this are quite possibly not on your keyboard at all.

Table A-2 has the most commonly used HTML entities. For the complete list, which includes many more international language characters, see www.webmonkey.com/reference/Special_Characters. You can also type in certain special characters using a non-English keyboard or pick international language characters from a utility program. See page 133 for more information about these options.

Table A-2. HTML character entities.

Character	Name of Character	What to Type
<	Less than	<
>	Greater than	>
&	Ampersand	&
"	Quotation mark	"
©	Copyright	©
®	Registered trademark	®
¢	Cent sign	¢
£	Pound sterling	£
¥	Yen sign	¥
€	Euro sign	€ (but € is better supported)
°	Degree sign	°
±	Plus or minus	±
÷	Division sign	÷
×	Multiply sign	×
□	Micro sign	µ
¼	Fraction one-quarter	¼
½	Fraction one-half	½
¾	Fraction three-quarters	¾
¶	Paragraph sign	¶
§	Section sign	§
"	Left angle quote, guillemotleft	«

Character	Name of Character	What to Type
"	Right angle quote, guillemotright	»
i	Inverted exclamation	¡
ı	Inverted question mark	¿
Æ	Small ae diphthong (ligature)	æ
Ç	Small c, cedilla	ç
È	Small e, grave accent	è
É	Small e, acute accent	é
Ê	Small e, circumflex accent	ê
Ë	Small e, dieresis or umlaut mark	ë
Ö	Small o, dieresis or umlaut mark	ö
Ё	Capital E, acute accent	É

HTML Color Names

The HTML standard officially recognizes only 16 color names. Table A-3 lists them.

Table A-3. HTML color names.

Aqua	Navy
Black	Olive
Blue	Purple
Fuchsia	Red
Gray	Silver
Green	Teal
Lime	White
Maroon	Yellow

Although many browsers recognize more names, the safest option to get better colors is to use a *color code* (see page 150).

Useful Websites

Throughout this book, you learned about a number of great websites where you can download handy software or get valuable information. Odds are, you'll want to revisit some of those sites to keep honing your web skills (or just to get free stuff). To save you the effort of leafing through hundreds of pages, this appendix provides those links, grouped by chapter.

To avoid carpal tunnel syndrome, you don't need to painstakingly type these URLs into your browser. Instead, use the online version of this appendix located on the Missing CD page at www.missingmanuals.com/cds/caw3. That way, once you find a site you want to visit, you're just a click away. In addition, check this page for late-breaking changes (like URLs that have moved to another location).

Chapter Links

The following tables list the links found in each chapter. Each table lists the links in the same order they occurred in the text. You'll find all kinds of links here. Some point to useful tutorial sites and articles, others to web curiosities, and still more to handy free tools or downloadable pictures and media.

Chapter 1. Preparing for the Web

Description	URL
The history of the Internet	www.isoc.org/internet/history www.walthowe.com/navnet/history.html
Browser usage statistics	http://en.wikipedia.org/wiki/Usage_share_of_web_browsers/

Description	URL
Internet Explorer (browser)	www.microsoft.com/ie
Firefox (browser)	www.mozilla.org/firefox
Google Chrome (browser)	www.google.com/chrome
Safari (browser)	www.apple.com/safari
Opera (browser)	www.opera.com
Summary of Mac browsers	www.knutson.de/mac/www/browsers.html
One of many free blogging services (see the Chapter 13 link list for more)	www.blogger.com
What not to do in a web page	www.angelfire.com/super/badwebs
The ultimate examples of bad web design	www.worstotheweb.com

Chapter 2. Creating Your First Page

Description	URL
Flash games	www.ferryhalim.com/orisinal
HTML validator	http://validator.w3.org

Chapter 3. Putting Your Page on the Web

Description	URL
Country top-level domains	http://tinyurl.com/rnlmf
Smartdots (free subdomain names)	www.smartdots.com
Google Sites (simplified page design)	http://sites.google.com
Brinkster (web host)	www.brinkster.com
DreamHost (web host)	www.dreamhost.com
GoDaddy (web host)	www.godaddy.com
HostGo (web host)	www.hostgo.com
Insider Hosting (web host)	www.insiderhosting.com
OCS Solutions	www.ocssolutions.com
Pair Networks (web host)	www.pair.com
Sonic.net (web host)	www.sonic.net
WebHostingTalk (discussion board)	http://tinyurl.com/5zffwp
TinyURL (URL-shrinking service)	http://tinyurl.com
Directory of free web hosts	www.free-webhosts.com

Chapter 4. Power Tools

Description	URL
Shareware programs	www.download.com
List of web page editors	http://en.wikipedia.org/wiki/Comparison_of_HTML_editors
Blue Griffon (web page editor)	www.bluegriffon.org
Amaya (web page editor)	www.w3.org/Amaya
HTML-Kit (web page editor)	www.htmlkit.com
CoffeeCup (web page editor)	www.coffeecup.com/free-editor
Expression Web (trial software)	www.microsoft.com/Expression/try-it
Dreamweaver (trial software)	www.adobe.com/products/dreamweaver

Chapter 5. Text Elements

Description	URL
Special characters in HTML	www.webmonkey.com/reference/Special_Characters

Chapter 6. Style Sheets

Description	URL
CSS compatibility tables for different browsers	www.quirksmode.org/css/contents.html http://caniuse.com
Online color pickers	www.colorpicker.com www.colorschemer.com/online.html
Font support on different operating systems	http://tinyurl.com/cr9oyx http://tinyurl.com/325f9qs
Embedded font support in different browsers	http://tinyurl.com/cluxzz
Ready-made font kits	www.fontsquirrel.com/fontface
Google Font Directory	http://code.google.com/webfonts
Font kit generator	www.fontsquirrel.com/fontface/generator

Chapter 7. Adding Graphics

Description	URL
Using SVG graphics	http://tinyurl.com/2gy2vyg
Using alpha blending in Photoshop	http://tinyurl.com/ypf78g
Free image editors	www.gimp.org (all platforms) www.getpaint.net (Windows-only) www.aviary.com (online)
Free backgrounds	www.grsites.com/textures www.backgroundsarchive.com

Description	URL
Google image search (pictures aren't necessarily free to use)	http://images.google.com
Stock.XCHNG (free pictures)	http://sxc.hu
Flickr (advanced search page)	www.flickr.com/search/advanced
Overview of places to find free pictures	http://tinyurl.com/49yquv3
Commercial picture sites	www.istockphoto.com www.fotolia.com www.dreamstime.com
Microsoft Office clip art	http://office.microsoft.com/images

Chapter 8. Linking Pages

Description	URL
A tool for encoding mail-to message text	http://meyerweb.com/eric/tools/dencoder
Link checker	http://validator.w3.org/checklink

Chapter 9. Page Layout

Description	URL
Web Developer add-in for Firefox and Chrome	http://chrisspederick.com/work/web-developer
A huge catalog of style sheet layout examples	www.csszengarden.com
Create a sticky footer	www.cssstickyfooter.com
An example of a fluid layout	www.dccdesign.co.uk
An example of an elastic layout	http://tinyurl.com/67v6hb6
An article about fixed, fluid, and elastic layout	http://tinyurl.com/q54tnl

Chapter 10. Multipart Pages

Description	URL
EBook about Expression Web page templates	http://tinyurl.com/9whbc7

Chapter 11. Introducing Your Site to the World

Description	URL
The Open Directory Project	http://dmoz.org http://dmoz.org/add.html (submission rules) http://dmoz.org/guidelines/ (editor guidelines)
Google Directory	http://directory.google.com
Yahoo Directory	http://dir.yahoo.com

Description	URL
Yahoo Directory submission guidelines	http://tinyurl.com/47quxhk www.apromotionguide.com/yahoo.html (unofficial) https://ecom.yahoo.com/dir/submit/intro (Yahoo Directory Submit)
Google site submission	www.google.com/addurl
Bing site submission	www.bing.com/webmaster/SubmitSitePage.aspx
How Google PageRank works	www.markhorrell.com/seo/pagerank.html http://en.wikipedia.org/wiki/PageRank
Google Toolbar	www.google.com/toolbar (download) http://tinyurl.com/64bjmtd (enabled the PageRank feature)
Search engine industry news	www.webmasterworld.com www.searchengineland.com
List of web robots	www.robotstxt.org
Wayback Machine (archived web pages)	www.archive.org
Google Webmaster Tools	www.google.com/webmasters/tools www.google.com/webmasters/tools/removals (remove a site)
Google Analytics	www.google.com/analytics

Chapter 12. Website Promotion

Description	URL
Google Places	http://places.google.com/business
Google AdWords	http://adwords.google.com
AdWords poetry	www.iterature.com/adwords
Form submission script	www.freicontactform.com/free.php
Form submission services	www.emailmeform.com www.123contactform.com
Twitter	http://twitter.com
Most popular Twitterers	http://twitaholic.com/top100/followers
Google Groups	http://groups.google.com http://groups.google.com/groups/groups/overview.html
Facebook fan page examples	www.facebook.com/kristof www.facebook.com/benjerry http://tinyurl.com/6amf47k

Description	URL
Facebook	www.facebook.com/pages/create.php (create a fan page) http://tinyurl.com/4z9wzlz (get a vanity URL) http://developers.facebook.com/docs/plugins (get a Like box)

Chapter 13. Blogs

Description	URL
Search engine for blogs	http://technorati.com
Blog examples	http://stefaniewildertaylor.com www.rottengods.com www.schneier.com http://wilheaton.typepad.com http://thesartorialist.blogspot.com http://roseandsnail.com
Group blog examples	http://lifehacker.com www.dailykos.com www.thehuffingtonpost.com
Online feed readers	www.google.com/reader www.newsgator.com
Windows feed reader	www.feeddemon.com
Mac feed reader	http://netnewswireapp.com
Browser feed readers	www.microsoft.com/windows/rss (IE) www.apple.com/safari/features.html (Safari)
Self-hosted blogs	http://wordpress.org www.movabletype.org
Hosted blogs	www.blogger.com http://wordpress.com www.typepad.com
Content Management Systems	http://drupal.org www.joomla.org
Information for Blogger	www.bloggerforum.com (discussion forum) http://help.blogger.com (official guide) http://tinyurl.com/n2zsbl (using a custom domain) http://tinyurl.com/4tbxvl8 (hiding the NavBar) http://tinyurl.com/295vg5 (template tag reference) http://tinyurl.com/3n4hv (creating a BlogThis bookmark)

Chapter 14. Making Money with Your Site

Description	URL
Google AdSense	www.google.com/adsense (sign up) www.google.com/services/adsense_tour www.google.com/adsense/taxinfo www.google.com/adsense/policies
Amazon Associates	http://affiliate-program.amazon.com http://tinyurl.com/46cqc84 (banners)
CafePress	www.cafepress.com
Zazzle	www.zazzle.com
PayPal	www.paypal.com www.paypal.com/chargeback www.paypal.com/SellerProtection www.paypal.com/fees

Chapter 15. JavaScript: Adding Interactivity

Description	URL
JavaScript properties	www.w3schools.com/html/dom/dom_reference.asp
JavaScript events	www.w3schools.com/jsref
JavaScript samples	www.dynamicdrive.com www.huddletogether.com/projects/lightbox2 www.javascriptkit.com/cutpastejava.shtml www.webmonkey.com/tutorial/JavaScript_Tutorial
JavaScript libraries	http://jquery.com http://mootools.net www.prototypejs.org http://dojotoolkit.org
Widget libraries	www.widgetbox.com www.google.com/ig/directory?synd=open www.widgipedia.com .

Chapter 16. Fancy Buttons and Menus

Description	URL
Button image generator	www.buttongenerator.com www.grsites.com/button
CSS button generator	http://css-tricks.com/examples/ButtonMaker
Slashdot menu	www.dynamicdrive.com/dynamicindex1/slashdot.htm

Chapter 17. Audio and Video

Description	URL
Flash games	www.ferryhalim.com/orisinal
WAV/MP3 editors	www.goldwave.com www.fleximusic.com
iTunes	www.apple.com/itunes
Flash MP3 player	www.premiumbeat.com/flash_music_players
Flash background music loops	www.flashkit.com/loops
Flash video players	http://flowplayer.org www.premiumbeat.com/flash_video_players
Expression Encoder	http://tinyurl.com/243b7sr
Making HTML5 video support multiple browsers	http://camendesign.com/code/video_forEverybody
YouTube	www.youtube.com http://tinyurl.com/66onkub (help topics) http://tinyurl.com/2kxnv3 (how to configure a video window)

Appendix A. HTML Quick Reference

Description	URL
JavaScript workaround for HTML5 on Internet Explorer	http://tinyurl.com/nlcjxm
HTML5 specification	http://dev.w3.org/html5/markup
HTML5 outliner	http://gsnedders.html5.org/outliner
Special characters in HTML	www.webmonkey.com/reference/Special_Characters

Index

Symbols

& (ampersand) in HTML, 30

@font-face feature, 165

;(semicolon) in HTML, 30

A

About Me section (Blogger), 370

absolute positioning (CSS), 259–261

absolute sizing (fonts), 162, 249

absolute URLs, 215

absolute value (position property), 255

access level (Google Groups), 332

active pseudo-class (CSS), 223, 472, 473

ad banners, 71

add-ons, browser, 11

address bar, browser, 23

Adobe Dreamweaver. *See* Dreamweaver, Adobe

AdSense, Google. *See* Google AdSense

advertisements, 384–385

affiliate programs, 385

alert() function (JavaScript), 428

Allow & Block Ads tab (AdSense), 390

alpha blending, 192

alt attribute, 184

Amaya web page editor, 82–83

Amazon Associates

 associate links, generating, 405–410

 overview, 403–404

 signing up as, 404–406

anchor element (<a>), 213–214

animated GIFs, 490

anti-aliasing, 192

Aplus.Net web hosting, 68

apps, Facebook, 343–344

archiving Blogger posts, 368–369

<area> element, 227

argument, defined (JavaScript), 428, 436

arithmetic operators, 432

arrays, 480

artifacts, compression, 191

<artist> element (HTML), 497

art on the Internet, 209–211

associate links, generating (Amazon Associates), 405–410

attributes, HTML, 40

audio

 embedded, 494–495

 Flash MP3 players, 495–498

 Flashtrek loops, 500–502

 Premiumbeat player, 495–498

 and video files, 489

 Yahoo Media Player, 498–500

autoplay attribute (<embed> element), 494

AVI files, 489

B

bold () element (HTML), 27

background-color property (CSS), 149–150, 172–173

background images, 201–205

background-image style property, 202

background music, 492–495

background-position property, 203

background property, 472

background-repeat property, 203

backgrounds, applying to specific elements, 207–208

backslash (\) character, 222

badges, Facebook, 347–348

bandwidth

 calculating need for, 65

 defined, 64

binary files, 20

Bing, Microsoft, 294

- blended transparency (images), 192
<blockquote> element, 115–117
block elements (HTML), 107–108
block-level elements (HTML), 42–43
Blogger
 advantages of, 356
 comments, managing, 379–382
 creating blogs with, 358–363
 custom domain names for blogs, 372–374
 disadvantages of, 357
 emailing entries, 365
 formatted posts, creating, 363–364
 group blogging, 370
 managing blogs with Dashboard, 365–368
 profiles, configuring, 370–372
 templates, customizing. *See* templates, Blogger
 tweaking common settings, 368–370
blogs
 basics of, 350–351
 BlogThis tool, 382
 hazards of, 353
 hosting of, 356
 overview, 349
 popular blogs, 351–352
 promoting, 382
 syndication of, 353–357
 video blogging, 507
BlueGriffon web page editor, 81–83
<body> element (HTML), 32
<body> section, scripts and (HTML), 427
bold and italics formatting, 128–129
bookmarks
 browser, 321
 to fragments, 229–231
borders (CSS)
 around images, 195–196
 basics, 172
 border-style attribute, 466
 designing, 172–173
 using with tables, 173–174
bounce rates (site usage), 309–310
boxes
 fixed, 245–246
 floating, 244–246
**
 element (HTML)**, 28
Brinkster web hosting, 70–72
browsers
 analysis of URLs by, 52–54
 bookmarks, 321
 browser-based uploading, 75–76
 compatibility with CSS, 137
 ignoring HTML tags with, 30
 unsupported by JavaScript, 430
 viewing HTML pages with, 22
bulleted list item tag (HTML), 36–37
bullets in lists, 208
business accounts (PayPal), 413
business websites, 14–15
buttons. *See also* fancy buttons
 Buy Now buttons (PayPal), 415–417
 shopping cart buttons (PayPal), 421–422
-
- C**
- cache keyword**, 299
CafePress, 410
calling functions (JavaScript), 435
caniuse.com, 137
captions, adding to images, 199–201
CDATA section (HTML), 430–431
cell spans (HTML tables), 126–128
centering fixed-width layouts, 250
channels feature (AdSense), 394–395
character entities (HTML), 30, 130–132, 540–541
chargebacks (PayPal), 412
charmap utility, 133
cite attribute, 117–118
class=“review” attribute, 179
class rules (CSS), 179
class selectors (CSS), 145, 147–148
clear property, 197–198
client-side programming (JavaScript), 424–425
client-side scripts, 64
Code view, creating web pages in, 88–89
CoffeeCup Free HTML Editor, 84
collapsible menus, 476–481
collapsible pages, 450–453
colors
 cautious use of, 152
 color palette (AdSense), 393–394
 finding, 151
 of hyperlinks, 222–224
 names of (HTML), 542
 placing pictures on colored backgrounds, 191–194
 properties (CSS), 149–150
 specifying, 150–151
colspan attribute (tables), 126
columns
 sizing (tables), 263–264
 stretching height of (layouts), 254–256
comments
 CSS, 176
 HTML, 39
 JavaScript, 437
 managing in Blogger, 379–382
 in scripts, 462
 tab in Blogger, 366
compression of images, 188, 190–191
contact forms, HTML, 325

- container elements (HTML),** 28–29, 107
content attribute (meta elements), 288
content management systems (CMS), 356
Content Overview (Google Analytics), 312–313
contextual selectors, saving work with,
 180–181
controls
 defined (HTML forms), 453
 form, listed, 455–456
 playback, 506
conversion rates (website sales), 320
cookies, 305
cords attribute (<area> element), 227
Creative Commons licenses, 210–211
CSS (Cascading Style Sheets)
 absolute and relative positioning, combining,
 259–261
 advanced layouts, 257
 attaching to web pages, 139–143
 basics, 136
 borders. *See borders (CSS)*
 browser compatibility with, 137
 building (tutorial), 175–179
 cascading behavior of, 144–145
 class selectors, 147–148
 color, cautious use of, 152
 color properties, 149–150
 colors, finding, 151
 colors, specifying, 150–151
 comments, 176
 creating for entire websites, 182
 creating with web page editors, 142
 CSS-Tricks, 474–475
 display property, 451
 external style sheets, 144
 font properties. *See font properties (CSS)*
 id selectors, 148
 inline styles, 143–144
 internal style sheets, 142
 overview, 105–107
 re-designing page layouts with, 256–258
 rollover effects with (buttons), 469–472
 rules, anatomy of, 138–139
 rules, types of, 136–137
 spacing around elements, 155–156
 spacing properties and text alignment, 152
 style inheritance, 145–147
 text alignment, 154–156
 type selectors, 147–148
 white space, 156–157
currentDate variable example (JavaScript),
 433–434
cursor property, 478
custom domain names (blogs), 372–374
custom functions (JavaScript), 434
customizing Blogger templates. *See templates, Blogger*

D
Daily Kos, 370
Dashboard, Blogger, 365–368
Dashboard, Google Analytics
 Content Overview, 312–313
 graphs of site visits, 308
 Map Overlay, 310–311
 Site Usage, 309–310
 Traffic Sources Overview chart, 311
 view of, 307
dashes (-) in domain names, 54
declaring functions (JavaScript), 434–435
declaring variables (JavaScript), 431
definition lists (HTML), 121–122
deleting
 comments in Blogger, 379
 element, 129
designing borders (CSS), 172–173
Design tab (Blogger), 367, 374, 375
dictionary list (<dl>) element, 122–123
dictionary term (<dt>) element, 122–123
digital audio/video files, 489
directories, registering websites with, 290–294
display property (CSS), 451, 471, 476, 479
display style property, 453
<div> element
 line breaks and spaces added with, 476
 saving work with, 179–181
 structuring web pages with, 243
divisions and spans (text), 117–118
DNS (Domain Name Service) catalog, 52
doctype, 31–32
documents
 document.getElementById() method, 441
 document object, 441
 document object (JavaScript), 433
 document object model (DOM), 440
 document-relative links, 221
 document type definition (DTD), 31–32
 saving as web pages, 80
 structuring with sectioning roots, 537
domain forwarding, 58, 59–61
domain names
 choosing, 54–55
 custom for blogs, 372–374
 defined, 18, 50, 64
 free subdomain services, 61
 international, 56
 registering, 57–58
 top-level domains, 55
domain parking, 57, 59
domain tasting, 55

- donations**, 384
drawing programs vs. image editors, 193
Dreamstime, 211
Dreamweaver, Adobe
 defining websites in, 97–100
 overview, 84
 uploading websites in, 99–101
 using page templates with, 279–282
Drupal, 356
dynamic buttons, 466
Dynamic Drive, 460, 482
Dynamic HTML (DHTML)
 HTML objects and, 440–445
 overview, 440
-
- E**
- e-commerce websites**, 14–15
editable regions (page templates), 273
editing. *See also web page editors*
 Blogger posts, 366–367
 HTML in Blogger templates, 378–379
- editors**
 image, 193
 web page. *See web page editors*
- elastic layouts**, 257
- elements**
 HTML, 27–29, 41–44
 HTML Reference, 515–533
 semantic (HTML5), 535–541
- email**
 addresses, selecting for websites, 64, 65
 emailing blog entries (Blogger), 365
 newsletters, 325–327
- embedding**
 <embed> element (HTML), 492–494
 embedded fonts, 165–166
 multimedia, 487–489
 video clips, 504–505
 YouTube videos, 512–514
- emphasized text () element**, 128
- empty element syntax (HTML)**, 28
- end tags (HTML)**, 27
- EOT (Embedded OpenType) font files**, 166
- equal sign (=) in JavaScript**, 431
- errors, checking web pages for**, 45–48
- events (JavaScript)**, 446–449
- event websites**, 14
- Expression Web**
 creating button pictures with, 469
 defining websites in, 92–95
 FrontPage server extensions and, 66
 linking to pages in, 214
 uploading websites in, 94–96
 using page templates with, 279–282
- external links**, 215
external script files (JavaScript), 439–440
external style sheets (CSS), 136, 144
-
- F**
- Facebook**
 fan pages, adding tabs to, 343–347
 fan pages, creating, 339–343
 pages, promoting on websites, 347–348
- fancy buttons**
 button pictures, creating, 467–469
 creating, 465–467
 picture-less buttons, 474–475
 picture-with-text buttons, 472–473
 rollover effects with CSS, 469–472
- fancy menus**
 collapsible menus, 476–481
 overview, 475–476
 third-party menus, 481–486
- fan pages (Facebook)**
 adding tabs to, 343–347
 creating, 339–343
- favorite icons (favicons)**, 321–322
- FBML (Facebook Markup Language)**, 346
- FeedDemon**, 354
- feed readers**, 353–355
- <figure>/<figcaption> elements (HTML5), 201
- file formats for graphics**, 188–192
- file names in URL paths**, 51
- files**
 browser-based uploading, 75–76
 FTP access, 72–75
- Firefox browser**, 11
- fixed boxes**, 245–246
- fixed content**, 273
- fixed value (position property)**, 255
- fixed-width layouts**, 247–250
- fixed-width pages**, 240–242
- Flash**
 Flash CS5: The Missing Manual, 505
 Kit, 500
 MP3 players, 495–498
 plug-in, 490–491
 video, 505–506
- Flashtrek loops**, 500–502
- FlexiMusic**, 495
- Flickr**, 210
- float attribute**, 398
- floating boxes**, 244–246
- float property**, 196–197
- flow elements**, 108
- Flowplayer**, 505
- fluid (liquid) layouts**, 257

- folders and relative links**
 moving into parent folders, 220–221
 moving into subfolders, 219–221
 moving to root folders, 221–222
 overview, 217–219
- folders and relative URLs, 222**
- font properties (CSS)**
 basics, 157–158
 creating/converting font formats, 170–171
 embedded fonts, 165–166
 font-family attribute, 159–160
 font kits, 167–170
 selecting fonts, 160–161
 sizing fonts, 161–165
 web formats for fonts, 166–167
- Font Squirrel website, 167, 170**
- for loops, 480**
- formatted posts, creating (Blogger), 363–364**
- formatting documents, 104–105**
- forms, HTML.** *See* interactive forms (HTML)
- form submission services, 326**
- forward slash (/) character, 222**
- Fotolia, 211**
- fragments, 229–231**
 URL, 51
- freeware, defined, 80**
- FrontPage server extensions, 66**
- FTP (File Transfer Protocol)**
 defined, 64
 uploading files using, 72–75
- functions (JavaScript), 428, 434–438**
-
- G**
- gadgets, changing/rearranging in templates (Blogger), 375–378**
- getElementById() function, 445**
- getElementById() method, 441–442**
- GIF format**
 animated, 490
 compression of, 191
 defined, 188
 saving as, 193
 when to use, 189–190
- GIMP image editor, 193**
- GoldWave, 495**
- Google accounts, 358**
- Google AdSense**
 ads, creating, 390–395
 AdSense for Search, 399
 ads, placing in web pages, 396–398
 basics of, 385–387
 Google-powered searches and, 399–403
 interface, 389–391
 rules, 389
 signing up for, 387–389
 targeted ads, creating, 399
- Google AdWords, 319–321**
- Google Analytics**
 basics of, 303–305
 Dashboard. *See* Dashboard, Google Analytics
 signing up for, 305–307
 web traffic, examining, 307–309
- Google Chrome, 11**
- Google Groups**
 basics of, 330–331
 creating, 330–335
 managing, 336–338
 participating in, 335–337
 restricting membership in, 337
- Google Places, 317–319**
- Google-powered searches, 399–403**
- Google Reader, 354**
- Google search engine, 294**
- Google Sites service, 62**
- Google Webmaster Tools, 299–303**
- graphical web page editors, 78**
- graphics.** *See also* images
 applying backgrounds to specific elements, 207–208
 bullets in lists, 208
 file formats for, 188–192
 free art, finding, 209–211
 graphical text, 204–206
 programs, 193
- graphs of site visits (Google Analytics), 308**
- GraveContainer rule, 261**
- group blogging, 370**
- groups, Internet, 329–330**
- growIncrement variable (text), 444–445**
-
- H**
- <head> element (HTML), 32**
- <head> section, scripts and (HTML), 427**
- headings (text), 112–113**
- height attribute (HTML), 186–187**
- height of columns, stretching (layouts), 254–256**
- height property (<tr attribute>), 265**
- hexadecimal color values (CSS), 150–151**
- hidden attribute (<embed element>), 494**
- horizontal lines (<hr> element), 113–114**
- hosting**
 blogs, 356
 multimedia, 487–489
- hotspots, adding to pictures, 226–229**
- hover pseudo-class (CSS), 223, 472**
- href (hypertext reference) attribute, 140, 213**
- HTML (HyperText Markup Language).** *See also* text
 basic elements for text, 107–109
 character entities, 540–541

- code views, 86
 color names, 542
 comments, 39
 documents, adding content to, 34–35
 documents, skeleton of, 32–34
 document type definition (DTD), 31–32
Dynamic HTML (DHTML). See Dynamic HTML (DHTML)
 editing in Blogger templates, 378–379
 editors. *See* web page editors
 elements, 27–29
 elements for tables, 124–127
 export features, 80
 file extensions, 23
 files, anatomy of, 20–23
 files, creating, 22–24
 <html> element, 32
 HTML5, 25–26
 HTML5 doctype, 32
 HTML5, JavaScript and, 426
 HTML5, new elements in, 118
 HTML5, video in, 506–507
 HTML-Kit, 83
 interactive forms. *See* interactive forms
 list elements. *See* list elements (HTML)
 nesting elements, 29–30
 objects, 440–445, 448–449
 pictures, linking to documents, 39–41
 scripts and, 429
 tags, ignoring with browsers, 30
 tags, overview of, 26–27
 text, structuring, 35–39
 tree model of elements, 38
 validating, 46–48
 versions of, 24–26
 viewing web pages, 22–23
- HTML Quick Reference**
- elements, 515–533
 - HTML5 semantic elements, 535–541
- HTTP (HyperText Transfer Protocol), 50**
- hyperlinks**
- anchor element (<a>), 213–214
 - associate links (Amazon Associates), 405–410
 - bookmarks, 229–231
 - broken, 231–232
 - colors and underlining of, 222–224
 - defined, 18
 - descriptive text in, 289
 - for opening page in new browser window, 217
 - hyperlink viewer (Expression Web), 233
 - image links/image maps, 226–229
 - internal and external links, 214–216
 - link checkers, 233–236
 - linking to different file types, 226
- mailto links, 224–225
 redirects, 236
 relative links and folders, 217–223
-
- I**
- <i> (italics) element, 117
id selectors (CSS), 148
Illustrator, Adobe, 193
images. *See also* graphics
- adding captions to, 199–201
 - alternate text and, 184–185, 288–289
 - background, 201–205
 - basics of, 183
 - borders around, 195–196
 - compression of, 190–191
 - file sizes for, 190
 - image ads (AdSense), 391
 - image editors, 193
 - image links/image maps, 226–229
 - image rollover events, 449–451
 - element (HTML), 39, 184
 - inline images in text, 194–195
 - picture sizes, 186–188
 - preloading, 472
 - wrapping text around, 196–200
- Include command (SSIs), 270–272**
- Info tabs (Facebook), 343–344**
- inheritance, style (CSS), 145–147**
- inline elements (HTML), 43–44, 107**
- inline formatting, 128**
- inline images in text, 194–195**
- inline styles (CSS), 137, 143–144**
- innerHTML property, 442**
- <input> element (forms), 455
- interactive forms (HTML)**
- basic form, example of, 456–459
 - controls, listed, 455–456
 - <form> element, 454–455
 - overview, 453–454
- internal links, 214–215**
- internal style sheets (CSS), 137, 142**
- international domain names, 56**
- Internet Explorer**
- declining popularity of, 12
 - external link quirk in, 216
 - overview, 10
 - security warnings and JavaScript, 429
- Internet vs. intranet, 53**
- intranets, 53**
- IP addresses, 52–53, 304**
- iStockPhoto, 211**
- italics and bold formatting, 128–129**
-
- J**
- JavaScript**

- browsers not supporting, 430
 categories of, 460
 collapsible pages, 450–453
Dynamic HTML and. See Dynamic HTML (DHTML)
 events, 446–449
 external script files, 439–440
 finding scripts online, 459–464
 functions, 434–438
 image rollover events, 449–451
 interactive forms. *See interactive forms*
 libraries, 462–463
 overview, 423–424
`<script>` element, 427–431
 scripting languages, 425–426
 server-side/client-side programming, 424–425
 spaces and line breaks in, 430
 variables, 431–435
- Joomla**, 356
- JPEG format**
 compression of, 191
 defined, 188
 saving as, 193
 when to use, 189–190
- justify setting (CSS text)**, 154
-
- K**
- keywords**
 for blog posts, 362
 keyword meta elements, 288
 keyword sizing of fonts, 162–163
- kits, font**, 167–170
-
- L**
- large font sizes**, 162
- layering (page layouts)**, 258–260
- layout, page**. *See page layouts*
- `` (bulleted list) element (HTML), 36–37
- libraries, JavaScript**, 462–463
- Like buttons (Facebook)**, 347
- line breaks**
`
` element (HTML), 28, 110–112
 in HTML documents, 37
 in JavaScript, 430
- linking. See also hyperlinks**
 link pseudo-class (CSS), 223
 link checkers, 233–236
 link rot, 231–232
 link unit ads (AdSense), 391
 link validators, 232
 to multimedia, 487–489
 pictures to HTML documents, 39–41
 to video clips, 504–505
- list elements (HTML)**
 definition lists, 121–122
- nesting lists, 122–124
 ordered lists, 119
 overview, 119–120
 unsorted lists, 121
- lists**
 bullets in, 208
`list-style-image` property, 208
- live bookmarks (Firefox)**, 355–356
- local() function (CSS)**, 168
- log analysis programs**, 303
- logical elements/documents**, 104–105
- long tail technology**, 352
- loop attribute (`<embed>` element)**, 494
- lossy/lossless compression**, 188, 191
- lowercase in domain names**, 54
-
- M**
- mailto links**, 224–225
- MakeVisible() function**, 479
- Map Overlay (Google Analytics)**, 310–311
- maps, image**, 226–229
- margin property (CSS)**, 155, 173–174, 472
- Mark of the Web comment**, 429
- markup, HTML**, 27
- masking URLs**, 61
- max-width/min-width properties (layouts)**, 253–254
- medium font sizes**, 162
- menus. See fancy menus**
- Merchant Services tab (PayPal)**, 414
- metadata, defined**, 93
- meta tags (elements)**, 287–288
- Meyer, Eric**, 257
- Microsoft Expression Web**, 84–85
- Microsoft Silverlight**, 505
- MIDI (synthesized music) files**, 489
- mobile-only style sheets**, 242
- moderated messages (Google Groups)**, 337
- moderating comments (Blogger)**, 380–382
- modifying variables**, 432–433
- Monetize tab (Blogger)**, 367
- monochrome color (CSS)**, 152
- monospaced fonts**, 114
- Movable Type**, 356
- MOV files**, 489
- MP3 files**, 489
- MPEG files**, 489
- multimedia**
 background music, 492–495
 defined, 487
`<embed>` element, 492–493
 files, types of, 489–491
 Flash MP3 players, 495–498
 Flashtak loops, 500–502
 linking/embedding/hosting, 487–489
 Premiumbeat player, 495–498

- scenarios, 492
 - video clips. *See* video clips
 - Yahoo Media Player, 498–500
 - Yahoo Player, 498–500
 - multipart pages**
 - on big websites, 269
 - overview, 267–269
 - page templates. *See* page templates
 - server-side includes (SSIs), 269–272
 - multiple column layouts**
 - column height, stretching, 254–256
 - fixed-width layouts, 247–250
 - max-width/min-width properties, 253–254
 - overview, 246–247
 - reformatting layouts with CSS, 256–258
 - resizable layouts, creating, 250–253
 - MultiplyNumbers() function (JavaScript)**, 438
 - music, background, 492–495
-
- N**
- name attribute (meta elements)**, 288
 - NavBar, Blogger**, 378
 - negative margins (CSS)**, 156
 - nesting elements (HTML)**, 29–30
 - nesting lists (HTML)**, 122–124
 - NetNewsWire**, 354
 - news aggregators**, 353
 - NewsGator**, 354
 - newsletters, email**, 325–327
 - noindex/nofollow values (robots meta element)**, 297
 - nonbreaking spaces ()**, 110
 - non-English language encoding**, 133–134
 - Number() function (JavaScript)**, 457
-
- O**
- objects, HTML**, 440–445
 - Office Online (Microsoft)**, 211
 - onmouseover event (JavaScript)**, 446
 - Open Directory Project (ODP)**, 290–293
 - Opera browser**, 11
 - ordered lists (HTML)**, 119
 - OTF (OpenType PostScript) fonts**, 166
-
- P**
- <p> element (HTML)**, 33
 - padding property (CSS)**, 173–174, 254, 472
 - page layouts**
 - absolute and relative positioning, 259–261
 - advanced CSS layouts, 257
 - creating with multiple columns. *See* multiple column layouts
 - layering, 258–260
 - screen space, 239–243
 - sizing tables, 261–265
 - style-based layout, 243–246
 - PageRank (Google)**, 295–296
 - page templates**
 - anatomy of, 276–279
 - basics of, 273–274
 - creating, 275–276
 - defined, 268
 - overview, 272–273
 - putting into action, 279–282
 - Paint.NET**, 193
 - paragraph (<p>) element (HTML)**, 33, 109
 - paragraph.style.fontSize property**, 445
 - parameters, example of (JavaScript)**, 436
 - parental blogs**, 352
 - paths**
 - <path> element (HTML), 497
 - URL, 50–51
 - pay-for-content sites**, 385
 - PayPal merchant tools**
 - chargebacks, 412
 - overview, 411–412
 - shopping cart service, 419–420
 - signing up, 412–413
 - withdrawing money, 421
 - percentage sizing**, 163–164, 254
 - percentage widths (table columns)**, 264
 - Performance Reports tab (AdSense)**, 390–391
 - personal accounts (PayPal)**, 412
 - personal websites**, 13–14
 - photography, stock**, 210
 - Photoshop, Adobe**, 193
 - phrasing elements**, 108–109
 - pictures**
 - adding hotspots to, 226–227
 - linking to HTML documents, 39–41
 - picture-less buttons, 474–475
 - picture-with-text buttons, 472–473
 - placing on colored backgrounds, 191–194
 - sizes of, 186–188
 - pixels**
 - defined, 241
 - sizing fonts with, 165
 - <playlist> element (HTML)**, 497
 - plug-ins for web pages**, 45
 - PNG format**
 - compression of, 191
 - defined, 188
 - saving as, 193
 - when to use, 189–190
 - pop-up blockers**, 217
 - position property (<div element>)**, 245, 255
 - Posting tab (Blogger)**, 366
 - <pre> element (CSS)**, 157
 - preferences, font**, 158
 - preformatted text (<pre> element)**, 114–116
 - preloading images**, 472

premier accounts (PayPal), 413
 Premiumbeat players, 495–498, 505
 presentational elements, 104
 product links (Amazon Associates), 406–407
 Products & Services tab (PayPal), 414
 profiles, configuring in Blogger, 370–372
 Programming the Mobile Web (O'Reilly), 243
 programs, graphics, 193
 progressive downloading (YouTube), 507
 progressive enhancement of web pages, 453
 promoting websites
 favorite icons (favicons), 321–322
 Google AdWords, 319–321
 Google Places, 317–319
 via newsgroups/articles/bulletin boards,
 316–317
 return visitors, attracting, 320–321

promotion-oriented websites, 14
properties

 CSS rules, 138
 HTML object, 443
proportional fonts, 114
proportional sizing of web pages, 240
protocols, defined (URLs), 50
 .psd format (Photoshop), 193
pseudo-classes (CSS), 223
puts, defined, 99

Q

<q> element (quotes), 117
 query strings, URL, 51
 QuirksMode, 31–32, 137
 quotation marks ("") in JavaScript, 431, 447
 quotes (<blockquote> element), 115–117

R

rankings, search engine, 296–297
 Rbrowser, 73–74
 recursion, validating links with, 234–235
 redirects (hyperlinks), 236
 reformatting templates (Blogger), 374–376
registering
 domain names, 57–58
 websites with directories, 290–294
 websites with search engines, 294–295
relative links, folders and, 217–223
relative positioning (CSS), 259–261
relative sizing of fonts, 164–165
relative URLs, 215–216
 resizable layouts, 250–253
resolution basics, 241
résumé websites, 14
return command (JavaScript), 438–439
return visitors, attracting, 320–321
RGB color values (CSS), 151

rich media ads (AdSense), 391
 rich programming environments, 425
 right position property, 250
 robot search programs, 297
 robots meta element, 297
 rollover buttons, defined, 466
 rollover effects with CSS (buttons), 469–472
 root-relative links, 221
 Rotten Gods blog, 352
 rowspan attribute (tables), 127
 rows, sizing (tables), 265
rules, CSS
 anatomy of, 138–139
 style, 136

S

Safari browser, 11
SafeSearch (Google), 401
sandboxed code (JavaScript), 424
sans-serif fonts, 159
Sartorialist blog, 352
saving
 documents as HTML, 80
 image file formats, 193
 work, with contextual selectors, 180–181
 work, with <div> element, 179–181
Schneier, Bruce (blog), 352
Schuman, Scott (blog), 352
screen-reading programs, 185
screen space, 239–243
scripting languages, 425–426
scripts
 defined, 426
 HTML and, 429
 HTML objects in, 443–446
 JavaScript, finding online, 459–464
 <script> element (JavaScript), 427–431

seamless frames (HTML5), 268

search engines
 Google Webmaster Tools, 299–303
 hiding from, 297–299
 PageRank (Google), 295
 registering sites with, 294–295
 rising up in rankings, 296–297
 search engine optimization (SEO), 296
 tricking, avoidance of, 289
 workings of, 285

searching
 Google-powered searches, 399–403
sectioning roots, structuring documents with,
 537
 <select> element (HTML), 455
selectors (CSS rules), 138
self-hosted blogs, 356
semantic elements (HTML5), 104, 535–541

serif fonts, 159
 server-side includes (SSIs), 268, 269–272
 server-side programming, 424–425
 server-side scripts, 66, 326
 setTimeout() function (JavaScript), 445
 Settings tab (Blogger), 367
 Shamshiri, Fariborz, 352
 shape attribute (<area> element), 227–228
 SharePoint Designer, 84
 shareware, defined, 80
 shopping cart service (PayPal), 419–420
 ShowAlertBox() function (JavaScript), 436–438
Silverlight, Microsoft, 505
 site management tools, 232–234
 sitemaps, 301
 site statistics, 65
 size variable (text), 444
sizing
 columns in tables, 263–264
 fonts, 161–165
 pictures, 186–188
 rows in tables, 265
 tables, 261–265
 slash (/) character, 28, 51, 184
 Slashdot menu, 482–486
 small print (<small>) element, 129
 smartdots subdomain service, 61
Smashing CSS: Professional Techniques for Modern Layout, 257
 software, blogging, 356
 source pages, 217
 spaces
 in HTML documents, 37
 in JavaScript, 430
 spacing around elements (CSS), 155–156
 spam, comment (blogs), 379
 element (text), 118, 200, 476
 spanning (HTML tables), 126–128
 special characters (HTML), 130–132
 split window editors, 79
 src attributes, 184, 439, 494
 standalone elements, 107
 standalone elements (HTML), 28–29
 standards mode (browsers), 31
 start tags (HTML), 27
 Static FBML tabs (Facebook), 346
 statistics, site, 65
 Stats tab (Blogger), 367
 Sticky Footers, 257
 sticky sites, 315
 Stock.XCHNG, 210–211
 string, defined, 428
 element (HTML), 128
 structuring HTML documents, 36, 104

style-based layout, 243–246
 style sheets. *See* CSS (Cascading Style Sheets)
 subdomain services, 61
 subscriptions to Google Groups, 332
 subscripts (<sub>) element, 129
 superscript (<sup>) element, 129
 SVG (Scalable Vector Graphics) font format, 166
 SVG (Scalable Vector Graphics) format, 188
 syndication (blogs), 353–356

T

tables
 basic table elements, 124–127
 cell spans, 126–128
 sizing, 261–265
 using borders with, 173–174
tags (HTML)
 ignoring with browsers, 30
 overview of, 26–27
 targeted ads, creating (AdSense), 399
 target web pages, 217
 Technorati, 351
templates, Blogger
 changing/rearranging gadgets in, 375–378
 customizing, 373
 editing HTML in, 378–379
 reformatting, 374–376
templates, page. *See* page templates
text. *See also* HTML (HyperText Markup Language)
 alignment and spacing (CSS), 152–155
 alternate for images, 184–185, 288–289
 basic HTML elements for, 107–109
 best practices, 134
 bold and italics formatting, 128–129
 CSS. *See* CSS (Cascading Style Sheets)
 descriptive in hyperlinks, 289
 divisions and spans, 117–118
 graphical, 204–206
 headings, 112–113
 horizontal lines, 113–114
 HTML vs. word processing, 103–104
 inline formatting, 128
 inline images in, 194–195
 line breaks, 110–112
 links (Amazon Associates), 408–409
 logical structure vs. physical formatting, 104–105
 preformatted, 114–116
 quotes, 115–117
 special characters, 130–132
 structuring (HTML), 35–39
 <textarea> element (HTML), 455

- text-decoration property, 453, 472
 TextEdit, 20, 77–79, 134
 text editors, 20, 78–79
 Unicode encoding, 133–134
 wrapping around images, 196–200
- <th> (table heading) element, 124**
- third-party menus, 481–486**
- three-panel style sheet layout, 254**
- tiling background images, 201**
- TinyURLs, 68
- <title> element (HTML), 33–34, 497**
- title text (web pages), 285–286**
- ToggleVisibility() function, 479**
- topical websites, 14**
- top-level domain (URLs), 55**
- <tr> (table row) element, 124**
- tracking website visitors**
 Google Analytics. *See* Google Analytics
 overview, 302–303
- Traffic Sources Overview chart (Google Analytics), 311**
- traffic virus, 320**
- transferring files. *See* files**
- transparent pixels, 192**
- tree model of HTML elements, 38**
- TTF (TrueType) fonts, 166**
- tutorial: building style sheets, 175–179**
- Twitter, 327–329
- type attribute, 120–121**
- TypePad, 356
- type selectors (CSS)**
 defined, 147–148
 rules for, 139
 vs. contextual selectors, 180–181
- typographic elements, 104**
-
- U**
- tag (HTML), 36**
- underline (<u>) element, 129**
- underlining hyperlinks, 222–224**
- Unicode encoding, 133–134**
- universal standards, HTML and, 36**
- unordered lists (HTML), 36, 121**
- updates to websites, 321–322**
- uploading**
 videos to YouTube, 509–513
 websites in Dreamweaver, 99–101
 websites in Expression Web, 94–96
- URLs (Uniform Resource Locators)**
 absolute, 215
 basics of, 49–51
 browser analysis of, 52–54
 masking, 61
 relative, 215–216
 rules for writing URLs in anchor elements, 222
- TinyURLs, 68
 url() function (CSS), 168
- usemap attribute (element), 228**
- Usenet, 329**
- User-Agent part (robots.txt file), 298**
- UTF-8, 134**
-
- V**
- validation tools, 46–48**
- value attribute (<input> elements), 455**
- values (CSS rules), 138**
- variables (JavaScript), 431–435**
- vector drawings, 188**
- video clips**
 audio files and, 489
 Flash video, 505–506
 in HTML5, 506–507
 linking to and embedding, 504–505
 overview, 502
 preparing, 502–504
 Silverlight plug-in, 505
 uploading to YouTube. *See* YouTube
 <video> element (HTML5), 506–507
- View Blog tab (Blogger), 367**
- viewport window, 255**
- visited pseudo-class (CSS), 223, 472**
- visitor tracking. *See* tracking website visitors**
- visual composer (Blogger), 364**
- volume attribute (<embed> element), 494**
-
- W**
- W3C validator tool, 46–48**
- Wall tab (Facebook), 343–344**
- watermarks, 203**
- WAV files, 489**
- web browser basics, 8**
- web communities**
 email newsletters and, 325–327
 Facebook. *See* Facebook
 Google Groups. *See* Google Groups
 tools for creating, 323–324
 transforming websites into, 322–323
 Twitter, 327–329
- web design tools, 18**
- web formats for fonts, 166–167**
- web hosting**
 Aplus.Net walkthrough, 68–70
 assessing requirements for, 62–64
 basics of, 12–13
 Brinkster walkthrough, 70–72
 checklist, 65–66
 free hosts, limitations of, 71–72
 reviewing companies, 67–73
 WebHostingTalk, 67
 workings of, 49–53

- web logs.** *See blogs*
- web page editors**
- Amaya, 82–83
 - BlueGriffon, 81–83
 - CoffeeCup Free HTML Editor, 84
 - creating style sheets with, 142
 - finding free, 80–81
 - HTML-Kit, 83
 - launching, 86
 - multiple views, 86–88
 - professional, 84–85
 - types of, 79
- web pages**
- attaching CSS style sheets to, 139–143
 - checking for errors, 45–48
 - creating in Code view, 88–89
 - creating in WYSIWYG view, 89–92
 - defined, 18
 - descriptions summarizing individual, 287–288
 - faulty appearance of, 24
 - linking. *See hyperlinks*
 - links for opening in new browser window, 217
 - multipart. *See multipart pages*
 - overview, 19
 - permanence of, 299
 - placing ads in (AdSense), 396–398
 - structuring with <div> element, 243
 - title text, importance of, 285–286
- web servers**
- defined, 8
 - logs, 303
 - overview, 12–13
- websites**
- components of, 17–18
 - creating style sheets for entire, 182
 - defining in Dreamweaver, 97–100
 - defining in Expression Web, 92–95
 - designing, 16–17
 - lifespan of, 15
 - management of, 232–234
 - managing, 91–92
 - plan for promoting, 284–285. *See also* promoting websites
 - promoting Facebook pages on, 347–348
 - registering with directories, 290–294
 - making search-engine friendly, 285–289
 - Site Usage (Google Analytics), 309–310
 - tracking visitors to, 302–303
 - types of, 13–15
 - uploading in Dreamweaver, 99–101
 - uploading in Expression Web, 94–96
- websites, for downloading**
- Amaya web page editor, 83
 - art and photographs, 210–211
- audio conversion shareware, 495
 - blogging software, 356
 - BlueGriffon, 81
 - button-making tools, 468
 - CoffeeCup Free HTML Editor, 84
 - Dreamweaver, 85
 - Expression Web, 85
 - Firefox browser, 11
 - Flash Kit loops, 500
 - Flowplayer, 505
 - font kits, 167
 - free link checker, 234
 - Google Chrome, 11
 - HTML editors, 80
 - HTML-Kit, 83
 - image editors, 193
 - JavaScript, 459–460
 - listed by chapter/topic, 543–550
 - Opera, 11
 - Premiumbeat player, 495–498
 - Slashdot menu, 482
 - web widgets, 463
 - Yahoo Media Player, 498–500
- websites, for further information**
- advanced CSS layouts, 257
 - alpha blending, 192
 - Amazon Associates, 404
 - badly designed websites, 16
 - Blogger help, 357
 - Blogger template codes, 379
 - browser support of fonts, 166
 - color picking, 151
 - contact form scripts, 326
 - CSS-Tricks, 475
 - Facebook promotional options, 347–348
 - feed readers, 354
 - fonts, 161
 - form-creation services, 326
 - free web hosts, 72
 - Google AdSense, 387
 - Google AdWords, 319
 - Google Analytics, 305
 - Google Groups, 330
 - Google Places sign-up, 318
 - Google Sites service, 62
 - Google Webmaster Tools, 299
 - history of the Internet, 8
 - HTML element properties, 442
 - HTML object events, 448
 - JavaScript libraries, 463
 - listed by chapter/topic, 543–550
 - Mac browsers, 11
 - odd domain name choices, 56
 - Open Directory Project (ODP), 290
 - popular blog sites, 352

Premiumbeat players, 496
Rbrowser, 73
registering with search engines, 294
search engine optimization (SEO), 296
search engine robots, 298
Slashdot menu, 482
special characters, 132
Sticky Footers, 257
SVG in Internet Explorer, 188
Technorati, 351
Twitter, 327
using page templates in Expression Web, 282
video coding, 506
W3C validator tool, 46
WebHostingTalk, 67
Yahoo directory, 293–294
web space. *See also web hosting*
assessing need for, 62–64, 65
defined, 18
Wheaton, Wil (blog), 352
white space (CSS), 156–157
width attribute (HTML), 186–187
width property (columns), 263
Wilder-Taylor, Stefanie, 352
WOFF (Web Open Font Format), 166–167
WordPress, 356–357
World Wide Web
 overview of, 7–8
 vs. Internet, 8
wrapping. *See text wrapping*
wrapping text around images, 196–200
WYSIWYG
 creating web pages in, 89–92
 editors, 79
 views, 86

X

XHTML (Extensible HyperText Markup Language), 25–26
x-y coordinates (image maps), 228

Y

Yahoo
 directory, 293–294
 Media Player, 498–500
 search engine, 294
YouTube
 background, 507–508
 preparing videos, 508–509
 reasons for using, 507
 signing up with, 508–509
 uploading videos to, 509–513
 watching videos, 512–514

Z

Zazzle, 410
z-index, 258

Do

Colophon

Creating a Website: The Missing Manual, Third Edition, was composited in Adobe InDesign CS4 by Newgen North America. Rachel Monaghan and Teresa Elsey provided quality control and Dan Fauxsmith provided production assistance.

The cover of this book is based on a series design originally created by David Freedman and modified by Mike Kohnke, Karen Montgomery, and Fitch (www.fitch.com). Back cover design, dog illustration, and color selection by Fitch.

David Futato designed the interior layout, based on a series design by Phil Simpson. The text font is Adobe Minion, the heading font is Adobe Formata Condensed, and the code font is LucasFont's TheSansMonoCondensed. The illustrations that appear in the book were produced by Robert Romano using Adobe Photoshop and Illustrator CS5.