2ND EDITION

sitepoint

HTML UTOPIA: DESIGNING WITHOUT TABLES USING CSS

BY RACHEL ANDREW & DAN SHAFER

THE ULTIMATE BEGINNER'S GUIDE TO CSS

HTML Utopia: Designing Without Tables Using CSS, 2nd Edition

Thank you for downloading this four-chapter sample of Rachel Andrew's and Dan Shafer's book, *HTML Utopia: Designing Without Tables Using CSS, 2nd Edition*, published by SitePoint.

This excerpt includes the Summary of Contents, Information about the Authors, Editors and SitePoint, Table of Contents, Preface, four chapters of the book, and the index.

We hope you find this information useful in evaluating this book.

For more information or to order, visit sitepoint.com

Summary of Contents of this Excerpt

Summary of Additional Book Contents

HTML Utopia: Designing Without Tables Using CSS

by Dan Shafer

and Rachel Andrew

HTML Utopia: Designing Without Tables Using CSS

by Dan Shafer and Rachel Andrew

Copyright © 2006 SitePoint Pty. Ltd.

Technical Director: Kevin Yank
Expert Reviewer: Richard Rutter
Managing Editor: Simon Mackie
Technical Editor: Craig Anderson
Printing History:
First Edition: May 2003
Second Edition: April 2006

Editor: Georgina Laidlaw
Index Editor: Bill Johncocks
Cover Design: Jess Mason
Cover Layout: Alex Walker

Notice of Rights

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.

Published by SitePoint Pty. Ltd.

424 Smith Street Collingwood
VIC Australia 3066.
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 0-9752402-7-7
Printed and bound in the United States of America

About the Authors

Dan Shafer is a highly respected web design consultant. He cut his teeth as the first webmaster and Director of Technology at Salon.com, then spent almost five years as the Master Builder in CNET's Builder.com division.

Dan gained widespread recognition as a respected commentator on the web design scene when he hosted the annual Builder.com Live! conference in New Orleans. He has designed and built more than 100 web sites and is regarded as an expert in web user experience design and implementation.

The author of more than 50 previous titles on computers and technology, Dan lives in Monterey, California, with his wife of almost 25 years, Carolyn, and their Shiitzu dog, Albert Einstein.

Rachel Andrew is web developer and director of web solutions provider edgeofmyseat.com. When not writing code, she writes *about* writing code and is the coauthor of several books promoting the practical usage of web standards alongside other everyday tools and technologies. Rachel takes a common sense, real world approach to web standards, with her writing and teaching being based on the experiences she has in her own company every day.

Rachel lives in the UK with her partner Drew and daughter Bethany. When not working, they can often be found wandering around the English countryside hunting for geocaches and nice pubs that serve Sunday lunch and a good beer.

About the Expert Reviewer

Richard Rutter lives and works in Brighton, UK, where he is co-founder and Production Director for web consultancy Clearleft.[1] Richard has been designing and developing web sites for nigh on ten years and regularly harps on about web standards, accessibility, and mountain biking on his weblog.[2]

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals.

Visit http://www.sitepoint.com/ to access our books, newsletters, articles, and community forums.

[1] http://www.clearleft.com
[2] http://www.clagnut.com

This book is dedicated to One Mind, in the knowing that It is all there is.

—Dan Shafer

Table of Contents

Order the print version of this book to get all 500+ pages!

Order the print version of this book to get all 500+ pages!

 Order the print version of this book to get all 500+ pages!

Order the print version of this book to get all 500+ pages!

Order the print version of this book to get all 500+ pages!

Order the print version of this book to get all 500+ pages!

Order the print version of this book to get all 500+ pages!

Preface

I've been around the Web for a while now—some might say I've been here from the beginning. And one thing that always bothered me about the Web was its inherent inability to disentangle content from presentation. The interconnectedness of it all meant that, to produce a web site, you needed not only to have something to say, and some graphical design skills to make the presentation of that message look good, but you also needed to be a bit of a programmer. Initially, this "programming" was a pretty lightweight task: HTML markup, when all is said and done, isn't really programming. Still, it's more than just writing words and using a word processor to format them, or conceptualizing a display for a page—digitally or otherwise.

It's no surprise, then, that designers who had clear ideas about how they wanted their web pages to look were frustrated by the need to create complex sets of deeply nested tables even to *approximate* their visions. As designers created increasingly complex ideas, and web browsers diverged further and further from even the merest semblance of compatibility, the Web threatened to collapse under its own weight. Serious designers began lobbying for a complete break from HTML to some new approach to the Web. Chaos reigned.

The Holy Grail of the Web, back then, was the notion that authors should write, designers should design (and code HTML), and programmers should … well … program. Those boundaries had not been clear in the first few years of the Web.

Then, along came Cascading Style Sheets (CSS), the subject of this book. The governing forces of the Web, through the World Wide Web Consortium, better known as the W3C,[1] addressed the problem with the proposal that we divide presentation instructions and the structural markup of content into two separate kinds of files.

Things haven't been the same since, thank goodness! Now we can (mostly) separate what we say from the way it's presented to the user in a browser. I wager that most of today's web developers are fairly comfortable with CSS, and would be no more likely to think of embedding presentational instructions in their HTML than they would to consider mixing 23 fonts on the same web or print page.

[1] http://www.w3.org/

Since CSS emerged, dozens of books have been written about it. So when SitePoint approached me to write a CSS book, my first thought was, "Who needs another CSS book?" But as they began to reveal their vision to me, it made sense. It was indeed time for a book that took a different tack, based on the extensive experience of the web design community.

This book is different from the rest in two fundamental ways.

First, it focuses on the question of how to use CSS to accomplish some of the successes that web designers have spent significant amounts of time and energy to create using nested tables. In other words, this book doesn't try to start from scratch and become a CSS tutorial. Instead, it's a sort of introductory CSS design guide.

Second, it starts at the outside and works its way in. Most, if not all, other CSS books focus first on the little pieces: the attributes, values, and tags that comprise the syntax of CSS. They then explain how to put those pieces together into a web site.

This book begins by looking at how CSS should influence the overall design of a site, and how to put the CSS framework in place before you begin to deal with individual HTML elements and their styling.

Who Should Read this Book?

As I wrote this book, I had in mind web designers with at least a little experience building sites, who are curious about how CSS can help them become more effective designers. It's aimed at the beginner to intermediate designer. I'll assume a strong grasp of HTML, but that's about it.

What's in this Book?

Chapter 1: *Getting the Lay of the Land*

This first chapter serves as a brief introduction to CSS and the main concepts that we'll discuss throughout the rest of the book. If you haven't used CSS at all before, or you want to ensure that you understand the concepts fully before you get started, this chapter is a great place to start.

Order the print version of this book to get all 500+ pages!

Chapter 2: ***Putting CSS into Perspective***

In this chapter, we begin to use CSS in practical ways, and to discuss why we might want to use CSS rather than old-style methods like font tags for text styling, and tables for layout.

Chapter 3: ***Digging Below the Surface***

Picking up the pace, we start to look in some depth at how CSS works. Here, we consider the different ways in which we can add CSS to our documents, we discuss CSS selectors and rules, and we investigate the various shorthand properties that will help us streamline our CSS files. We'll also come to grips with the concept of inheritance. This chapter ensures that you understand the terminology and syntax we'll be using, which will make it easier for you to follow examples in this book and elsewhere.

Chapter 4: ***Validation and Backward Compatibility***

In this chapter, we discuss how we can validate our documents and style sheets to ensure that they comply with the published specifications. We also find out a bit about the practicalities of ensuring our sites' backward compatibility with older browsers or devices.

Chapter 5: ***Splashing Around a Bit of Color***

This chapter looks closely at the ways in which colors can be applied to text and other objects, as well as to page backgrounds. It will discuss how to describe colors, where to use them, and how to make them work together to achieve specific effects.

Chapter 6: ***Working with Fonts***

This chapter examines the question of how fonts can be used properly in CSS-based web design. After an explanation of how CSS deals with fonts at the most abstract level, we'll look at the use of standard and nonstandard fonts in web pages. Finally, we'll discuss some guidelines for the selection of font families and sizes for your page designs.

Chapter 7: ***Text Effects and the Cascade***

This chapter builds on Chapter 6, where we looked at text in terms of fonts and their related style properties. Here, we'll explore a range of other ways in which we can style text, and spend time looking at links and lists, in particular.

Chapter 8: ***Simple CSS Layout***

We start this chapter by creating a simple two-column layout. Along the way, we discover how to use absolute and relative positioning techniques in CSS

Order the print version of this book to get all 500+ pages!

layouts; how margins, padding, and borders work together; and how we can put all of these techniques into practice by creating a fully functional two-column layout.

Chapter 9: *Three-column Layouts*
Out first task in this chapter is to add a third column to the layout we created in Chapter 8. We then discuss the issues that arise when we want to add a footer that runs along the bottom of a multiple-column layout like ours. Along the way, we'll find out how to use the `float` property to create multi-column layouts, and how to create full-length columns using CSS. We'll also consider some of the issues that surround these types of layouts.

Chapter 10: *Fixed-width Layouts*
In this last chapter, we'll create a fixed-width layout that's centered in the user's browser window. As we progress, we'll look at techniques for styling data tables effectively, and discuss one method by which you can enable your users to choose a different layout if they find your fixed-width layout difficult to read.

Appendix A: *CSS Miscellany*
This appendix provides a brief description of some of the more obscure parts of CSS that weren't covered in detail earlier in the book, including the "at-rules" and aural style sheets. It also introduces the concept of DHTML as a launching point for further reading.

Appendix B: *CSS Color Reference*
This appendix provides a comprehensive list of all (official and unofficial) color names in CSS, along with their hexadecimal and RGB equivalent values.

Appendix C: *CSS Property Reference*
This sizeable appendix contains a complete reference to all CSS properties at the time of writing. It includes a practical example for each property (when appropriate) and gives an indication of the level of support browsers provide for that property.

Bibliography
The Recommended Resources listed here include books and web sites. The bibliography is by no means exhaustive; it's more of a list of our own favorite references—resources that we, personally, have found helpful over the years—than a reference to every resource on the topic.

Order the print version of this book to get all 500+ pages!

The Book's Web Site

Located at http://www.sitepoint.com/books/css2/, the web site supporting this book will give you access to the following facilities:

The Code Archive

As you progress through the text, you'll note a number of references to the code archive. This is a downloadable ZIP archive that contains complete code for all the examples presented in the book. It also includes a copy of the *Footbag Freaks* web site,[2] which we use as an example throughout the book. You can get it from http://www.sitepoint.com/books/css2/code.php on the book's web site.

Updates and Errata

No book is perfect, and we expect that watchful readers will be able to spot at least one or two mistakes before the end of this one. The Errata page, at http://www.sitepoint.com/books/css2/errata.php on the book's web site, will always have the latest information about known typographical and code errors, and necessary updates for new browser releases and versions of the CSS standard.

The SitePoint Forums

If you'd like to communicate with us or anyone else on the SitePoint publishing team about this book, you should join theSitePoint Forums.[3] In fact, you should join that community even if you *don't* want to talk to us, because there are a lot of fun and experienced web designers and developers hanging out there. It's a good way to learn new stuff, get questions answered (unless you really enjoy being on the phone with some company's tech support line for a couple of hours at a time), and just have fun.

The SitePoint Newsletters

In addition to books like this one, SitePoint offers free email newsletters.

[2] http://www.footbagfreaks.com/
[3] http://www.sitepointforums.com/

The SitePoint Tech Times covers the latest news, product releases, trends, tips, and techniques for all technical aspects of web development. The long-running *SitePoint Tribune* is a biweekly digest of the business and moneymaking aspects of the Web. Whether you're a freelance developer looking for tips to score that dream contract, or a marketing major striving to keep abreast of changes to the major search engines, this is the newsletter for you. *The SitePoint Design View* is a monthly compilation of the best in web design. From new CSS layout methods to subtle Photoshop techniques, SitePoint's chief designer shares his years of experience in its pages.

Browse the archives or sign up to any of SitePoint's free newsletters at http://www.sitepoint.com/newsletter/

Your Feedback

If you can't find your answer through the forums, or you wish to contact us for any other reason, the best place to write is `books@sitepoint.com`. We have a well-manned email support system set up to track your inquiries, and if our support staff is unable to answer your question, it comes straight to us. Suggestions for improvement—as well as notices of any mistakes you may find—are especially welcome.

Acknowledgements

First and foremost I must acknowledge the author of the original edition of this book, Dan Shafer, for the solid CSS tutorial that makes up the first half of the book. His original work still stood as an excellent introduction to the subject almost three years later, and updates were required simply due to the passing of time and the evolution of browsers since the first edition of this book was produced.

Thanks must also go to the team members at SitePoint—especially to Simon Mackie—for their expertise and support in guiding this book to completion. Also, thanks to expert reviewer Richard Rutter, who helped greatly in ensuring that outdated advice was excised from the original manuscript, and that I didn't add any inaccuracies of my own!

Finally, and as always, thanks to Drew and Bethany for putting up with me and supporting me through yet another book project. I love you both.

—Rachel Andrew

Order the print version of this book to get all 500+ pages!

1

Getting the Lay of the Land

We can look at Cascading Style Sheets (CSS) from a number of contextual perspectives. I prefer to view them as a correction to a fundamental mistake that was made at the beginning of Web Time, back in the old days of the early 1990s, when Tim Berners-Lee and the pioneering web builders first envisioned the beginnings of the Web.

What was that mistake?

To meet the requirements of the Web's initially limited purpose (its original intent was to allow a small number of nuclear physicists using disparate systems at various locations to share vital experimental data), it was not necessary to separate a page's content (the information contained in the document) from its presentation (the way that information is displayed). However, Berners-Lee didn't envision the massively popular, wildly commercialized, extensively morphed Web that emerged from his core ideas in the early 1990s—I doubt that anyone could have.

So, the mistake was a lack of foresight, rather than an oversight. But it was a mistake nonetheless.

CSS in Context

Almost as soon as the Web became popularized by the emergence of early graphical web browsers (such as the wildly popular Netscape Navigator), the designers of early web sites became aware of a problem. The method by which the web browser displayed information stored in HTML files was not within the designers' control. No, it was primarily the users who were in charge of how the web pages they visited would appear on their systems.

While there were many, including myself, who thought this was A Good Thing, designers were beside themselves with concern. From their perspective, this constituted a fundamental flaw. "Users don't know anything about good design," they argued. If the designers couldn't control with great accuracy things like colors, fonts, and the precise, pixel-level positioning of every design element on the web page, their creations could easily end up as ugly travesties in users' browsers. Most designers, accustomed to print and other fixed layouts that afforded them complete control over what the user saw, found ways to bend the Web to their will.

Lest I incur the ire of every designer reading this book, let me hasten to add that I don't think this was A Bad Thing. It is certainly the case that designers know more about how content should be displayed for users than do the users themselves. Things like spacing, color combinations, and other design elements affect readability and usability. My point has much less to do with who should have been in charge, than it does with the actions to which designers were more or less forced to resort in order to achieve at least some measure of control.

Soon, expert designers discovered that they could use tables to gain significant control over the presentation of content to users. By laying out tables within tables within tables, they could position quite precisely any design element that could be contained within a table cell. And that encompassed almost everything.

The first desktop publishing-style web page design tool, NetObjects Fusion, enabled designers to lay out pages with a high degree of precision. It generated complex, table-based HTML, which resulted in web pages that were as close as possible to the designer's original vision.

We never looked back.

But tables weren't intended to be used as layout tools, so while they were effective, they were also horribly inefficient. We'll explore some of the shortcomings and disadvantages of using tables for layout tasks a little later in this chapter; for now,

Order the print version of this book to get all 500+ pages!

just know that everyone, including the designers who used the techniques, understood pretty well how clumsy a solution they really were.

The Basic Purpose of CSS

After a brief series of skirmishes at the beginning of the Web's development, the question of who should control the overall appearance of a page or site ended with the designers as victors. Users, after all, care more about usability, accessibility, and convenience than the nitty-gritty details of design techniques.

Yet designers found themselves hard-pressed to identify very good, standards-compliant ways to provide their customers—and their customers' users—with great designs that were also effective and efficient. Thus, they were forced to rely largely on tables.

However, as time passed and the use of tables to lay out web pages became increasingly complex, even the design community became uneasy. Maintaining a web page that consists of a half-dozen or more deeply intertwined tables is a nightmare. Most designers prefer not to deal with code—even simple HTML markup—at such a level of detail.

Into the breach stepped the World Wide Web Consortium, better known as the W3C,[1] a body founded by Tim Berners-Lee to oversee the technical growth of the Web. They saw that separating the content of a site from its presentation (or appearance) would be the most logical solution. This would enable content experts—writers, artists, photographers, and programmers—to provide the "stuff" that people come to a site to see, read, or experience. It would also free the design experts—artists, graphic designers, and typographers—to determine a site's aesthetics independently of its content.

The result was CSS.

Why Most—but Not All—Tables Are Bad

Why is the table not suited to being a design mechanism? There are numerous reasons, but the ones we're most concerned with in this context are:

- They result in load times that are longer than necessary.

[1] http://www.w3.org/

- They encourage the use of inefficient "placeholder graphics" that further slow performance.
- Their maintenance can be a nightmare in which even minor changes break the entire layout.
- They can cause the page to become inaccessible to those who are not using a graphical web browser.

Tables Mean Long Load Times

Most people don't know that web browsers are deliberately designed to ensure that each table downloads as a single entity. None of the material that's contained in a table will be displayed until all the contents of that table are downloaded to the client machine and available for display.[2]

When the original, intended purpose of tables is taken into account, this makes sense. Tables were designed to display … well, tables of data. Each cell contained a value that was being compared to, or related with, the values of other cells in the table. Isolated bits of data appearing quasi-randomly would not do; the table was a single, integrated entity.

When designers began to rely on tables to contain all or most of the content of a web page, they were also saddled with the consequences of this design decision. In addition to the apparent delay that many users experience as a result of tables displaying all at once, the sheer volume of HTML code that's required to create web page layouts with nested tables can also add load time due to the increased page size. Table-based layouts almost certainly account for more user concern over long page-load times than any other single factor.

Avoiding this significant load time would obviously be A Good Thing.

Use of Transparent Images Slows us Down

Even when using tables as layout mechanisms, designers could not quite attain the detailed level of control they wanted over page design. Sometimes, for instance, a designer might need a bit more breathing room around one part of a table cell—something for which tables do not allow. This kind of precision was unachievable.

[2]Cascading Style Sheets Level 2 (CSS 2) includes a property called `table-layout` that alters this behavior, with several important caveats. Refer to Appendix C for details.

Order the print version of this book to get all 500+ pages!

Early on, someone came up with the notion of creating a `transparent.gif` image file—a tiny GIF image that had no visible content. By creating table cells that contained these transparent images, we could force extra vertical and horizontal "space" into tables whose cells were designed to remain in close proximity to one another.

The problem is that, given a table with dozens (or even hundreds) of these images, and depending on a variety of other factors, the performance impact of transparent GIFs on a web page can be significant. More importantly, though, this technique often restricts the page to a fixed pixel size, and clutters the page with images that are irrelevant to the meaning of the page content. This severely impacts the ability of users with disabilities to make sense of table-based sites, as we'll see later.

Maintaining Tables is a Nightmare

The third reason why most tables are bad is that maintaining a complex array of deeply nested tables is a nightmare. If you use tools such as Macromedia Dreamweaver or Adobe GoLive to manage your sites and their designs, generally you can ignore the messiness of the nested tables that make the design possible. But even these tools are not foolproof, and when they "mess up" (to use a highly technical term), amending the unsightly pages they create can be quite a challenge.

If you're like most designers, and you wouldn't be caught dead using an HTML-generating tool because you feel you gain more control and understanding if you hand-code everything, then you'll be familiar with the maintenance problem.

The difficulty arises because, by necessity, tables have a fairly complex set of tags—even if they aren't embedded within other tables. And when we have nested tables, well, we've got a clear case of the uglies, all right.

The situation is further complicated by the fact that, unlike programming editors, HTML editors generally do not force or support the clean indentation of code. So, finding the start and end points for a given table, row, or cell turns out to be what software folks call a "non-trivial task." While it's true that a competent HTML coder or designer could make this problem more tractable, it's never really solvable, no matter what we do.

Tables Cause Accessibility Issues

The fourth reason why tables are bad lies in the way non-graphical browsers—such as the screen readers used by many visually impaired users—read an HTML document. When a text-only device reads the content of a site, it starts at the top and works down the page line by line. When it comes to a table, it starts at the first (top-left) cell, then continues along the top row, then moves to the second row, and so on. In the case of a table that's used correctly, for tabular data, this is rarely a problem. However, where nested tables have been used to display chunks of text in the desired layout, that content can become nonsensical when read in this manner.

When it's Okay to Use a Table

There's one notable exception to the cardinal rule that Tables Are A Bad Thing.

If you have tabular data, and the appearance of that data is less important than its appropriate display in connection with other portions of the same data set, then a table is in order. If you have information that would best be displayed in a spreadsheet such as Excel, you have tabular data.

In general (though, undoubtedly, there are exceptions to this rule as well), this means that the use of tables should be confined to the presentation of numeric or textual data, not graphics, multimedia data types, forms, or any other interactive user interface components.

What is CSS, Really?

Now that we've established that an important role of CSS in designers' lives is to free us from the drudgery of using tables for page layout, let's take a look at what CSS really is.

The most important word in the label "Cascading Style Sheets" is the middle one: "style." "Cascading" becomes important only when we get into fairly complex style usage, while the word "sheet" is a tad misleading at times. So, even though we mean Cascading Style Sheets in the broadest and most accurate sense, we'll focus not on the cascading or sheet-like nature of these beasts, but on their role in determining the styles of our web pages and sites.

Order the print version of this book to get all 500+ pages!

Styles are defined as **rules**. These rules tell any web browser that understands them (i.e. any browser that supports CSS) how to display specific types of content structures when it encounters these structures in delivering a web page to a user. We call this visual display of a web page the way the browser **renders** the page.

To understand how styles affect the appearance of a web page, we need to understand what happens to a web page in the absence of any style rules.

Figure 1.1 shows how the browser displays a page when its author hasn't specified any style rules. Each browser has a default way of displaying web pages using its own internal style sheet. So, a first-level heading enclosed in `<h1>` and `</h1>` tags will be displayed using a relatively large font in black, because that's dictated by the browser's style sheet. The "default" font that's used may vary between browsers, and can be affected by user-defined settings as well.

Figure 1.1. Normal browser page display behavior

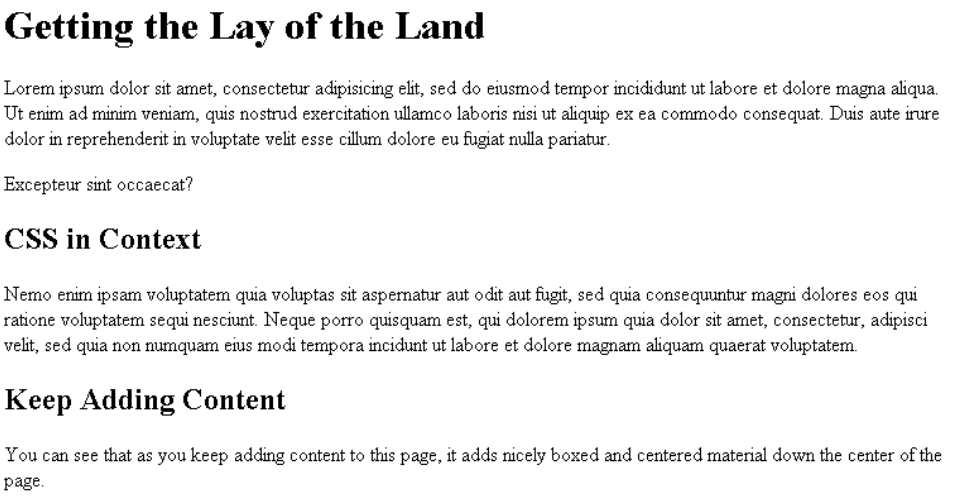


Figure 1.2. The browser displaying a page with a style rule in effect

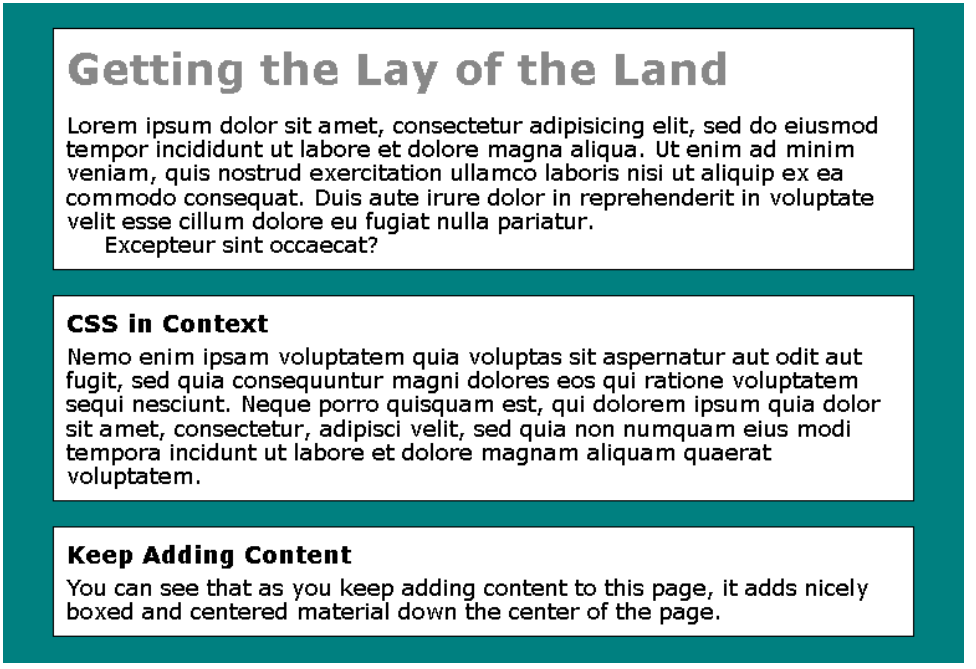


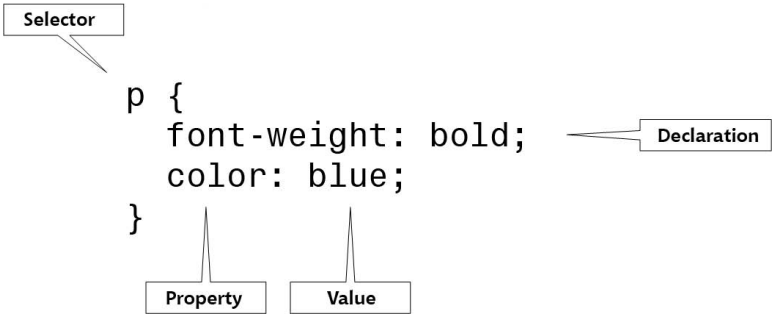
Figure 1.2 depicts what happens when the page's author defines style rules. An author-defined rule overrides the browser's own internal style sheet rule for that element, and the new style takes over. Even if the user has defined his or her own settings for this element, those wishes usually will not be honored (though there are some intriguing exceptions to this generality, which we'll discuss much later in this book).

Parts of a CSS Rule

Every style consists of one or more rules. Figure 1.3 shows a CSS rule with all the parts labeled.

Order the print version of this book to get all 500+ pages!

Figure 1.3. The parts of a CSS rule



Each rule has two parts:

1. a **selector** that defines the HTML element(s) to which the rule applies
2. a collection of one or more **declarations**, made up of a **property** and a **value**,[3] which describe the appearance of all the elements that match the selector

The property tells the browser which element is being defined. For example, `font-weight` tells the browser that this declaration defines the weight of the font. After the colon that separates the two parts of a declaration, we see a value that will be applied to that property. If a value of `bold` followed the `font-weight` property, it would make the weight of the font in that document bold. Each declaration must be followed by a semicolon, with one exception: the semicolon that follows the last property is optional and may be omitted. In this book, though, we'll always add the optional semicolon. I encourage you to adopt this habit, as it's much easier to train yourself always to add the semicolon than it is to remember when it is and is not required. This approach also makes it easier to add properties to an existing style rule.

Here are a few examples of increasingly complex CSS rules, with the parts identified so that you can fix this syntax clearly in your mind. This is the only real syntax issue you must understand in order to master CSS, so it's important!

[3]Many books and articles about CSS get confused when it comes to this terminology, using these terms interchangeably, or calling declarations "attributes." In this book, I used the W3C-endorsed terminology of "declarations," "properties," and "values." I reserve the name "attributes" for attributes of HTML tags.

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

The selector, `h1`, indicates that this rule applies to all `h1` headings in the document. The property that's being modified is `color`, which refers to the font color. The value we want the `color` property to take on is `red`. Chapter 5 and Chapter 6 explore fonts and coloring in CSS in greater detail.

```
p {
  font-size: small;
  color: green;
}
```

The selector, `p`, indicates the style rule should be applied to all paragraphs in the document. There are two declarations in the rule. The first, which sets the property `font-size`, sets the size of the font in all paragraphs in the document to `small`. See Chapter 3 for an explanation of this and other measurement issues in CSS. The second property, `color`, is set to `green`. The result of this rule is that all paragraphs in the document will appear in a green, "small" font.

```
p {
  font-family: 'New York', Times, serif;
}
```

Again, this rule deals with paragraphs, as is evidenced by the `p` selector. This time, the selector affects the font family that is used to display text. The new wrinkles in this example are that it includes a list of values for the `font-family` property, and one of those values is enclosed in quotation marks.

The `font-family` property is one of a handful of CSS properties to which you can assign a list of possible values, rather than a single, fixed value. When you use a list, commas must separate its individual members. In this case, the `font-family` value list tells the browser to use New York as the font if the user's machine has it installed. If not, it directs the browser to use Times. And if neither of these fonts is available on the user's system, the browser is told to default to the font used for serif type. This subject is covered in more depth in Chapter 6.

Whenever a value in a list includes spaces (as is the case with the font named "New York"), you must put that value into quotation marks. Many designers use single quotation marks for a number of reasons, not least of which is that they're slightly easier to type, but you can use either single or double quotation marks.

Order the print version of this book to get all 500+ pages!

Types of CSS Rules

We can categorize and think about CSS rules in several possible ways:

- First, we can think of the different types of properties that can be defined. For example, different properties affect the color of elements, their positions within the browser window, and so on.
- We can also consider the types of elements that can be affected using CSS, and specifically, how certain elements can be targeted.
- Finally, there is the issue of where the style rules are defined.

Let's take a brief look at each of these categorizations, so that you have a good overview of the organization of CSS rules before you embark on a detailed study of their use.

Which Properties can CSS Rules Affect?

CSS rules can include properties that affect virtually every aspect of the presentation of information on a web site. A complete reference to these properties is presented in Appendix C.

Which Elements can CSS Affect?

Stated another way, this question asks, "How, specifically, can a CSS rule target a piece of information on a web page for special presentation?" CSS allows the designer to affect all paragraphs, but how can you confine that impact to certain, specific paragraphs? Is this even possible?

The answer is "yes." Through various combinations of selector usage, the designer can become quite specific indeed about the circumstances under which a style rule is enforced. For example, you can assign rules so that they affect:

- all elements of a specific type
- all elements of a specific type that are assigned to a common group or class
- all elements of a specific type that are contained within other elements of a specific type

- all elements of a specific type that are both contained within another specific element type and assigned to a common group or class
- all elements of a specific type only when they come immediately after an element of some other type
- only a specific element of a specific type that is assigned a unique ID

Chapter 3 includes a detailed discussion of all the CSS selectors you can use to achieve these kinds of precision targeting.

Where can CSS Styles be Defined?

Finally, you can define CSS styles in any of three places:

- inside the HTML (such style declarations are called **inline declarations**)
- between `<style>` and `</style>` tags inside the `head` element (this is called an **embedded style sheet**)
- in an external CSS file, also called an **external style sheet**

Inline Declarations

You can style any element by listing style declarations inside that element's `style` attribute. These are referred to as inline declarations because they're defined inline as part of the document's HTML. You can assign a `style` attribute to almost all HTML elements. For example, to make a second-level heading within a document appear in red text and all capital letters, you could code a line like this:

```
<h2 style="color: red; text-transform: uppercase;">An Unusual
    Heading</h2>
```

If you follow the advice in this book, you won't use many inline declarations. As we'll see, separating content from presentation is one of the big advantages of CSS, and embedding styles directly in HTML tags defeats that purpose. Inline declarations are mainly useful for rapid prototyping—quickly applying style properties to a particular element to experiment with an effect before giving the properties a more permanent place in an embedded or external style sheet.

Order the print version of this book to get all 500+ pages!

Embedded CSS

Specifying style properties in an embedded style sheet is an approach that's often used by beginning web designers and those just learning the techniques involved in CSS design. It's not my favorite method, but it does have the virtue of being easy to deal with, so you'll see it used from time to time in this book.

To embed a style sheet in a web page, we place a `style` element in the head of the document's HTML and fill it with style rules, as shown here in bold:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>CSS Style Sheet Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
      h1, h2 {
        color: green;
      }
      h3 {
        color: blue;
      }
    </style>
  </head>
```

The CSS rules contained in the style block apply to all the designated parts of the current document. In this case, the first rule directs the browser to display all level one and two headings (`h1, h2`) in green. The second rule displays all level three headings (`h3`) in blue.

Notice that each rule starts on a new line, and each declaration within the rule appears indented within braces on its own line. Strictly speaking, this layout isn't required, but it's a good rule of thumb that improves the readability of your code, especially if you're used to the look of JavaScript code.

External CSS

Finally, you can define CSS rules in a file that's completely separate from the web page. You can link to this file by including a `link` element in the head of any web page on which you want to implement those styles.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>CSS Style Sheet Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <link rel="stylesheet" type="text/css" href="corpstyle.css" />
  </head>
```

In this example, the file `corpstyle.css` contains a set of styles that have been linked to this page. Here's what the contents of this file might look like:

File: **corpstyle.css**

```
h1, h2 {
  color: green;
}
h3 {
  color: blue;
}
```

This is my preferred way to use CSS, for a number of reasons.

First, this is the least "locked-in" of the three basic methods designers can use to insert styles into a web page. If you define an external style sheet file, you can apply it to as many pages of your site as you want, simply by linking to the style sheet from each page on which you want it used. Using external CSS also makes your site a lot easier to maintain: changing the appearance of an element that appears on every page of your site is a simple matter of modifying the shared `.css` file. If you use embedded or—worse yet—inline styles, you'll have to change every single page on which the element appears.

Second, external style sheets are treated as separate files by the browser. When the browser navigates to a new page that uses the same style sheet as a previous page, that external style sheet will not be downloaded again. Therefore, pages that use external styles are quicker to load.

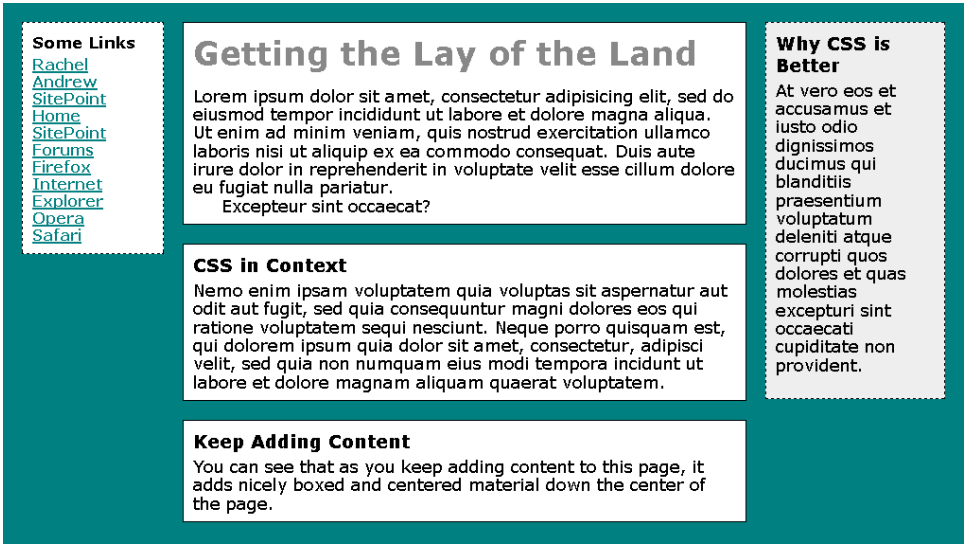
Last, but not least, external style sheets are simply more professional. By using them, you demonstrate an understanding of the importance of the separation of content from presentation, and you make it much easier to discuss your style sheets, share them with colleagues, analyze their effects, and work with them as if they were a serious part of the site's design, rather than an afterthought.

Order the print version of this book to get all 500+ pages!

A Simple Example

Now that you have a basic overview of what CSS is all about, why it exists, and why it's an important technique for web designers to adopt, where's the proof? Let's look at an example of a small but not overly simplistic web page (see Figure 1.4).

Figure 1.4. A sample web page demonstrating embedded styles



Here's the HTML that will produce that page if we use embedded CSS. Don't let the complexity of the code intimidate you—by the end of Chapter 3 you should be able to infer the meaning of most of it without my help. For now, you can download the code archive from the book's web site and marvel at the results in your browser. The file is called `ch1sample.html`.

File: **ch1sample.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Basic 3-Column Sample Page</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
```

```
body {
  background-color: teal;
  margin: 20px;
  padding: 0;
  font-size: 1.1em;
  font-family: Verdana, Arial, Helvetica, sans-serif;
}
h1 {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  margin: 0 0 15px 0;
  padding: 0;
  color: #888;
}
h2 {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  margin: 0 0 5px 0;
  padding: 0;
  font-size: 1.1em;
}
p {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  line-height: 1.1em;
  margin: 0 0 16px 0;
  padding: 0;
}
.content>p {
  margin: 0;
}
.content>p+p {
  text-indent: 30px;
}
a {
  color: teal;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-weight: 600;
}
a:link {
  color: teal;
}
a:visited {
  color: teal;
}
a:hover {
  background-color: #bbb;
}
/* All the content boxes belong to the content class. */
```


Order the print version of this book to get all 500+ pages!

```
    .content {
      position: relative;
      width: auto;
      min-width: 120px;
      margin: 0 210px 20px 170px;
      border: 1px solid black;
      background-color: white;
      padding: 10px;
      z-index: 3;
    }
    #navleft {
      position: absolute;
      width: 128px;
      top: 20px;
      left: 20px;
      font-size: 0.9em;
      border: 1px dashed black;
      background-color: white;
      padding: 10px;
      z-index: 2;
    }
    #navleft ul {
      list-style: none;
      margin: 0;
      padding: 0;
    }
    #navright {
      position: absolute;
      width: 168px;
      top: 20px;
      right: 20px;
      border: 1px dashed black;
      background-color: #eee;
      padding: 10px;
      z-index: 1;
    }
  </style>
</head>
<body>
  <div class="content">
    <h1>Getting the Lay of the Land</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
        sed do eiusmod tempor incididunt ut labore et dolore
        magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut aliquip ex ea
        commodo consequat. Duis aute irure dolor in
```

```
        reprehenderit in voluptate velit esse cillum dolore eu
        fugiat nulla pariatur.</p>
    <p>Excepteur sint occaecat?</p>
  </div>
  <div class="content">
    <h2>CSS in Context</h2>
      <p>Nemo enim ipsam voluptatem quia voluptas sit aspernatur
          aut odit aut  fugit, sed quia consequuntur magni
          dolores eos qui ratione voluptatem  sequi nesciunt.
          Neque porro quisquam est, qui dolorem ipsum quia dolor
          sit amet, consectetur, adipisci velit, sed quia non
          numquam eius modi tempora incidunt ut labore et dolore
          magnam aliquam quaerat voluptatem.</p>
  </div>
  <div class="content">
    <h2>Keep Adding Content</h2>
    <p>You can see that as you keep adding content to this page,
        it adds nicely boxed and centered material down the
        center of the page.</p>
  </div>
  <div id="navleft">
    <h2>Some Links</h2>
    <ul>
      <li><a href="http://www.rachelandrew.co.uk/"
          title="Rachel Andrew's personal site">Rachel
          Andrew</a></li>
      <li><a href="http://www.sitepoint.com/"
          title="SitePoint Home Base">SitePoint Home</a></li>
      <li><a href="http://www.sitepoint.com/forums"
          title="SitePoint Discussion Forums">SitePoint
          Forums</a></li>
      <li><a href="http://www.mozilla.org/firefox"
          title="Firefox at The Mozilla Foundation">Firefox</a>
          </li>
      <li><a href="http://www.microsoft.com/ie"
          title="Internet Explorer at Microsoft's Site">Internet
          Explorer</a>
      <li><a href="http://www.opera.com/"
          title="Opera Home Page">Opera</a></li>
      <li><a href="http://www.apple.com/safari"
          title="Safari on Apple's Web Site">Safari</a></li>
    </ul>
  </div>
  <div id="navright">
    <h2>Why CSS is Better</h2>
    <p>At vero eos et accusamus et iusto odio dignissimos
```


Order the print version of this book to get all 500+ pages!

```
          ducimus qui blanditiis praesentium voluptatum deleniti
          atque corrupti quos dolores  et quas molestias excepturi
          sint occaecati cupiditate non provident.</p>
    </div>
  </body>
</html>
```

Summary

You should now understand the historical and technological contexts in which CSS has emerged, the major problems it is designed to solve, and how it works on a superficial level. You also know why tables aren't suited to being used as a web page layout device, even though they have other perfectly valid uses.

In addition, you can identify both the parts of a CSS rule, and at least three ways in which these rules can be applied to your web pages.

Chapter 2 drills more deeply into the prospective issues surrounding CSS. It clears up some of the misconceptions you may have about this technology, and describes some of the important issues you'll have to take into consideration because of the way web browsers work (or don't) with CSS rules.

2 Putting CSS into Perspective

In Chapter 1, we took a 10,000-foot view of CSS. We began by looking at why using tables for web page layout is generally a bad idea. Then, we examined the types of CSS rules, and which aspects of a web page our style sheets could affect.

This chapter provides an overview of CSS's place in the web development cosmos. First, we'll discuss what CSS can and can't do for you. We'll spend a little time examining the advantages of CSS design, and see how using CSS can help you to create better sites by doing things that old-style tables and spacer GIFs can't do.

After a quick look at how CSS interacts with the ever-shifting world of web browsers, we'll discover how we can create CSS that accommodates those browsers that don't provide full support for CSS standards, either because they predate the standard, or they tried to support the standard but got it wrong.

What can CSS Do?

Recall from Chapter 1 that one of the key advantages of CSS is that it separates the *content* of a web site from its *appearance* or *presentation*. This separation is important because it allows us to create web sites that enable writers to create the *information* the web site is intended to convey, while leaving the *design* of the site—how it looks and how it behaves—to designers and programmers.

It follows, then, that CSS would be useful for defining the *appearance* of a site, but not necessarily for dictating its *behavior*.

However, like many such generalizations, this statement is true only most of the time. Why? Because the dividing line between appearance and behavior is necessarily fuzzy.

For example, as we'll see when we develop our layouts in the second part of this book, CSS can be used effectively to create context-sensitive menus, along with other elements of the interface with which your users will interact. You may be familiar with menu designs whose interactivity relies heavily on JavaScript, or some other scripting language, but we'll learn some techniques that avoid scripting, while allowing us to do some fairly creative things with navigation.

Later on, this book provides detailed instructions and examples of how you can alter the appearance of colors, fonts, text, and graphics using CSS; the rest of this section provides some ideas about the kinds of tasks for which you can use CSS. My intention here is less to teach you how to do these things than it is to whet your appetite and start you thinking about the possibilities …

Color and CSS

You can use style sheet rules to control the color of any HTML element that can be displayed in color. The most common elements for which you'll find yourself setting the color are:

- text
- headings (which are really a special form of text)
- page backgrounds
- background colors of text and headings

This may not seem like much, but knowing when and how to apply color to these elements—and, perhaps more importantly, how to combine the use of color in interconnected elements—can really expand your web design capabilities.

Order the print version of this book to get all 500+ pages!

Figure 2.1. Black-and-white version of fall holiday page

The simple act of changing the color of all the text on a page, then providing a colored background for that text, can turn a fairly ordinary-looking web page (Figure 2.1) into one that has a completely different feel to it. Figure 2.2 shows what the page in Figure 2.1 looks like if we simply choose colors appropriate to a holiday theme—yellow text on a black background. Figure 2.3 shows the opposite effect: black text on a yellow background. While you could argue that these alternative layouts aren't as readable as the black and white original in Figure 2.1, you'd have to admit that the two variations are more interesting to look at.

Figure 2.2. Yellow-on-black version of fall holiday page

Here's the style rule that creates the effect in Figure 2.2. As you can see, it's fairly straightforward, yet the result of its use is certainly dramatic.

```
body {
  color: yellow;
  background-color: black;
}
```

As we'll see in Chapter 5, naming the colors you want is just one of several ways to define color in CSS.

Here's the style rule that creates the effect in Figure 2.3. No surprises here: it's the opposite of the code that was used to generate the look in Figure 2.2.

```
body {
  color: black;
  background-color: yellow;
}
```

Figure 2.3. Black-on-yellow version of fall holiday page

Maybe you find the use of a starkly contrasting color for the entire background of a page a bit overwhelming. Figure 2.4 shows another variation on the text color theme. Here, we've provided yellow text on a black background only behind the headings on the page. The rest of the page's background color, and all non-heading text, remains unchanged from the original design in Figure 2.1.

Order the print version of this book to get all 500+ pages!

Figure 2.4. Yellow-on-black headings on fall holiday page

Here's the style rule that generates the heading effect shown in Figure 2.4.

```
h1, h2, h3, h4, h5, h6 {
  color: yellow;
  background-color: black;
}
```

Notice that we didn't have to do anything fancy, like put the headings inside `<div>` and `</div>` tags, or create a rectangular box around them. In the view of the web browser, the heading is a **block level element**, which occupies the full width of the space in which it resides, by default. So, if you give a heading a `background-color` property, that property will apply to the entire horizontal block that contains the heading.

CSS provides a range of other advantages to the color-conscious designer, but we'll leave those details to Chapter 5. Our purpose here is merely to touch upon the variety of things you can expect to accomplish using CSS.

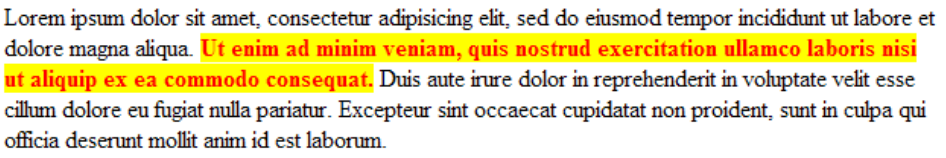
Fonts and CSS

In Chapter 1, we saw a number of examples that used fonts in CSS style rules. From that exposure, you're probably comfortable with defining the fonts in which you want the body text and headings of various levels to be displayed.

You can apply fonts to smaller amounts of text by enclosing that text within `<span>` and `</span>` tags (a subject we'll treat in detail in Chapter 9), then ap-

plying style properties to the `span`. You might use this approach, for example, to highlight a sentence in the middle of a paragraph, as shown in Figure 2.5.

Figure 2.5. Highlighting an important sentence



To do this, we simply need to wrap the sentence in `<span>` and `</span>` tags, then add a style rule for the new `span`. Note that these `span` elements should be used sparingly, and that there are a number of issues to consider before you apply these techniques—see Chapter 8 and Chapter 9 for all the details. Below is the HTML that was used to create this effect.

```
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
  do eiusmod tempor incididunt ut labore et dolore magna aliqua
  <span class="important">Ut enim ad  minim veniam, quis nostrud
  exercitation ullamco laboris nisi ut aliquip  ex ea commodo
  consequat.</span> Duis aute irure dolor in reprehenderit in
  voluptate velit esse cillum dolore eu fugiat nulla pariatur.
  Excepteur  sint occaecat cupidatat non proident, sunt in culpa
  qui officia  deserunt mollit anim id est laborum.</p>
```

You can target a particular `span` by adding an `id` or `class` attribute (we'll look at this in more detail in Chapter 3), then adding the `id` or `class` to the selector, as shown here:

```
.important {
  font-weight: bold;
  background-color: yellow;
  color: red;
}
```

One type of HTML text element to which it's sometimes quite useful to apply font rules is the list. We generally create lists in an effort to call specific attention to several items that are related to one another, and using a font style to set the list off even more clearly from the text can be a good technique. Figure 2.6 shows a list that has been set in a font that contrasts with the main text of the page,

Order the print version of this book to get all 500+ pages!

and is bold. The list stands out from the page, calling attention to itself as being particularly important.

Figure 2.6. Highlighting an important list

Once we've identified this list in HTML using an `id` attribute, we can style it by adding a rule to our style sheet.

```
<ul id="partylist">
  <li>children (at 7:30 p.m. in the downstairs kitchen)</li>
  <li>teens (at 9:30 p.m. in the youth room)</li>
  <li>adults (at 11:00 p.m. in the fellowship hall)</li>
</ul>
```

The rule now looks like this:

```
#partylist {
  font-family: 'Comic Sans MS', Arial, Helvetica, sans-serif;
  font-weight: bold;
  color: yellow;
  background-color: black;
}
```

Dynamic Pseudo-classes and CSS

One of the more interesting effects that you can create with CSS involves the use of the "hover" effect on text. By defining a CSS style rule that changes the appearance of text when the user pauses the cursor over that text, you can create an effect that looks a bit like animation.

Unfortunately, this effect works only on link text in Internet Explorer 6, although in other browsers—such as Firefox and Internet Explorer 7[1]—you can create this effect on other elements. You can use the `hover` **pseudo-class** to determine what will happen to a text link over which the user pauses the cursor, as shown here:

```
a:hover {
  background-color: blue;
  color: white;

}
```

Figure 2.7 shows what happens when the user positions the cursor over a link to which this style rule is applied. While you can't tell that the color of the text has changed, you can easily see that the text is larger than the other links around it.

Figure 2.7. Applying a dynamic pseudo-class to a hovered link

This effect feels a bit like an animated graphic in a menu where the buttons are programmed to change when the user's mouse hovers over them—it's a technique that we'll learn more about in Chapter 9.

Changing Text Size in `:hover` Styles

You may be tempted to change the size of the text in a link when the user hovers their mouse over it—it does make very obvious to the user which link

[1]At the time of writing, Internet Explorer 7 is still in beta testing, so no guarantees can be made of its final functionality.

Order the print version of this book to get all 500+ pages!

they currently have selected. However, this is generally considered bad practice, as changing the size of text in the middle of a document will typically move other elements of the document around, potentially confusing the user. It's much better to use background and font colors to make such distinctions.

Images and CSS

Images are placed on a web page using the HTML `<img />` tag. With CSS, we can only affect relatively minor aspects of an image's display, but that doesn't mean we can't control anything interesting.

Like any other object in a web page, an image can always be enclosed inside a `div` element and positioned arbitrarily on the page. We can also affect the border around an image, as well as its alignment, again by embedding the image in a `div` element, then using a style to alter the appearance of that containing `div`.

Figure 2.8 shows what would happen to an image placed alongside text on a page, in the absence of any CSS instructions. The image appears at the left edge of the page and it is aligned with one line of text, which shares its baseline with the bottom of the image. Subsequent lines of text appear below the image.

Figure 2.8. An image and text to which CSS styles haven't been applied

Subscribe to One of Our Journals Now... and Save!

Shown here is a selection of some of the many magazines and journals we have available for subscription as part of our annual fund-raising drive. As a matter of fact, we have just about every magazine and journal you could ever hope to find or subscribe to. If you already get enough magazines, we can arrange to renew your current subscription when it expires and still get credit for your purchase. We both win. You don't miss out on any issues of a proven favorite magazine, and Charities International gets the benefit of your contribution.

One thing for which CSS is particularly helpful is forcing text to flow gracefully around inline images. Using the `float` property (which is covered in detail in Chapter 8), you can "float" an image on a page in such a way that the text placed beside it will wrap around the image nicely. Figure 2.9 shows what happens if

we position the image using the `float` property. Note how the text flows smoothly around the side of, and then under, the image. This is almost certainly closer to the design effect we want than the example shown in Figure 2.8.

Figure 2.9. Positioning an image and text with help of `float`

To do this to all the images in your site, add the following rule to your style sheet:

```
img {
 float: left;
}
```

Multiple Style Sheets, Users, and CSS

It is possible to define more than one style sheet for a given web page or site; we'll look at how alternate style sheets can be used in the course of creating projects later in the book. Some modern browsers (such as Firefox and Opera) allow the user to select from additional style sheets if they have been created. These "alternate" style sheets can be used to display larger font sizes or higher contrast designs for users who have specific accessibility needs.

With a bit of scripting, you can automate that selection process and create an adaptable site that several different categories of users can experience appropri-

Order the print version of this book to get all 500+ pages!

ately. We won't be covering this kind of scripting in this book, but if you're interested, Paul Sowden's article, "Alternative Style: Working With Alternate Style Sheets",[2] on A List Apart is a great place to start.

Advantages of CSS Design

I've already touched on a number of the powerful features of, and reasons for, using CSS for site layout. In this section, I'll formalize those arguments and present them all in one place. Not only do I hope to convince you of the merits of CSS, but I aim to give you the tools to sell *others* on the technology.

In the cutthroat world of freelance web development, you will often be called upon to explain why you will do a better job than other developers bidding on the same project. If CSS layout is one of the tools in your web design arsenal, the sites you build will benefit from the advantages presented here. Many of these advantages go well beyond ease of development, and translate directly to extra value for your clients. Let them know about this—it just might make the difference between winning the contract and losing out to a designer who lives and breathes table-based design.

Increased Stylistic Control

Perhaps the major selling point of CSS is that it lets you control many aspects of the appearance of your site that simply cannot be controlled with pure HTML (for example, creating hover effects on links). For a complete reference to the style properties that can be controlled with CSS, see Appendix C.

In addition to the number of properties that it puts at your fingertips, CSS allows you to apply those properties to the available HTML page elements more uniformly than would be possible using other techniques. For instance, if you wanted to use HTML to put a visible border around part of the page, you'd need to use a table to do it, because pure HTML lets you add borders to tables only. Not only does CSS give you greater control over the look of the border (it can be solid, embossed, dotted, or dashed; thick or thin; any of a multitude of colors; etc.), it lets you add a border to *any* page element—not just tables. The design rationale behind CSS aims to give the designer as many options as possible, so, generally speaking, a property can be applied at any point at which, potentially, it could make sense to do so.

[2] http://www.alistapart.com/articles/alternate/

CSS simply has more properties that can be applied to more page elements than HTML has ever offered. If you had to choose between CSS and HTML as a means for specifying the design of your site, and your decision was based solely on which approach would afford you the most visual control, CSS would win outright. Despite this, it is common practice to use HTML for design wherever possible, and to resort to CSS whenever an effect is needed that HTML cannot produce. While the appearance of sites designed with this rationale is just as good as any others, by taking this approach to design, we miss out on all the other advantages of CSS.

Centralized Design Information

As I've already explained, the best way to use CSS in the design of a web site is to write one or more `.css` files to house all your style code, and then to link those files to the appropriate pages with the HTML `<link />` tag. This approach ensures that everything to do with the *look* of your site can be found in one place, and is not jumbled up with the *content* of your site.

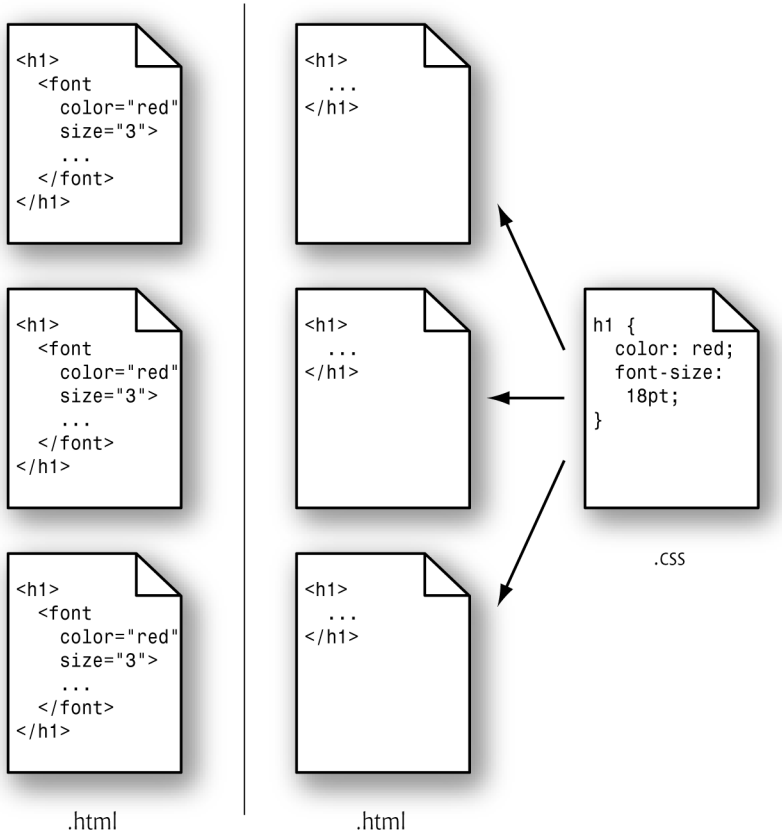
The idea is that you should be able to change the content of your site without affecting its look, and vice versa. In traditional web design, where HTML tags and attributes are used to specify the way things look in the browser, the code for these two aspects of your site are mixed together, so anyone who wants to modify one of these must understand both, or risk breaking one while making changes to the other. The look and the content of the site are said to be **coupled**.

This principle of keeping code that serves different purposes in different places is known in the programming world as **decoupling**. If a site's style and content are decoupled, a web designer can modify the look of the site by editing the `.css` file(s), while a content editor can add content to the site by editing the `.html` files.

Even more significant than facilitating organization and teamwork, this separation of code reduces code duplication. In HTML-based design, if you want the title of every article on your site to display in a large, red font, you have to put `<font>` and `</font>` tags around the text inside the relevant `h1` element on every one of your site's article pages. With CSS-based design, you can specify the font properties for every `h1` element in one place, which saves on typing. And, should you decide to change the appearance of these headings, you have only to modify the `.css` file instead of each and every `.html` file, which saves your sanity! These differences are illustrated in Figure 2.10.

Order the print version of this book to get all 500+ pages!

Figure 2.10. Centralizing design code with CSS



If you look closely at Figure 2.10, you'll see that, in addition to the organizational advantages described above, the browser has less code to download. On heavily designed sites, or sites with hundreds of pages or more, this reduced download time can have a significant impact both on the user experience, as well as your bandwidth costs.

Semantic Content Markup

When you use `.css` files to decouple the content from the appearance of your site, as I've just described, a curious thing begins to happen to your HTML. Because CSS affords you complete control over the appearance of page elements, you begin to choose tags because they describe the structure and meaning of

elements of the page, instead of how you want them to look. Stripped of most or all of the presentational information, your HTML code is free to reflect the **semantics** of your site's content.

There are a number of reasons why this is a desirable state of affairs, key among them the fact that decoupling content from design makes it very easy to find things when you're changing the content of your site. The easiest way to spot a CSS-based site is to use the View Source feature in your browser—if you can make sense of the code within ten seconds, chances are that you're not dealing with a site that uses table-based layout and other non-semantic HTML.

Your web site will be easier for potential visitors to find through search engines if it's marked up with semantic HTML, because the fewer presentational tags the search engine has to wade through to analyze your site, the easier it will be for it to index the content. As we'll see, CSS lets you control the position of an element in the browser window almost independently of its position in the HTML document. So, if you have a newsletter subscription form, or some other lengthy chunk of HTML that won't mean a whole lot to a search engine, feel free to move its code to the end of your HTML document and use CSS to ensure that it's displayed near the top of the browser window.

Increasingly supported by modern browsers is a feature of the HTML `link` element[3] that lets you restrict a linked style sheet so that it affects a page only when that page is displayed by a certain type of browser. For instance, you could link three `.css` files to a page: one that defined the appearance of the page on a desktop browser, another that dictated how the page will look when printed, and yet another that controlled the display on mobile devices such as Internet-connected Personal Digital Assistants (PDAs). Only by using semantic markup, and allowing the CSS to take care of the display properties, is this sort of content repurposing possible.

Last, but certainly not least, are the vast accessibility improvements that a site can gain by using semantic markup. We'll discuss these in detail in the next section.

Accessibility

Should you ever have the opportunity to observe a visually impaired individual browsing the Web, I highly recommend you do so. Alternatively, get yourself

[3]Specifically, the `media` attribute.

Order the print version of this book to get all 500+ pages!

some screen reader software, switch off your monitor, and see for yourself what it's like.

Web sites that use tables, images, and other non-semantic HTML for layout are extremely difficult for visually impaired people to use. Their screen reader software will typically read the page aloud, from top to bottom. It's not unusual for a modern, table-based web site to inflict 30 seconds or more of nonsense upon the user before the actual content begins. An example of some of what a screen reader would output for a table based site is shown below:

> Table with one column and five rows, Table with three columns and one row, Link, Graphic, slash logo underline main dot gif, Table end, Table with two columns and one row, Link, Graphic, slash nav underline about underline us dot gif, Link, Graphic, slash nav underline site underline map dot gif, Table end, Table end, Table with one column and twenty-six rows, Table with one column and seventeen rows …

Now, if you think that sounds mildly annoying, imagine having to listen to it for each and every page of the sites that you visit!

CSS-based design and semantic markup nearly eliminate this aural garbage, because they ensure that every tag in the document has a structural meaning that's significant to the viewer (or listener). An aural browser ignores the visual formatting properties defined in the CSS, so the user need not listen to them.

On a site that used semantic markup, for example, a visually impaired user would never have to wonder if a word was bold because it was more important, or just because it looked better that way. Elements that were displayed in bold for design reasons would have that property assigned using CSS, and the aural browser would never mention it. Elements that needed additional impact or emphasis would be marked up using the semantically meaningful `strong` and `em` elements, which are displayed, by default, as bold and italic text in visual browsers, yet also convey meaning to a screen reader user, as they tell the device to emphasize the phrase.

A complete set of guidelines exists for developers who are interested in making their sites more accessible for users with disabilities. The Web Content Accessibility Guidelines 1.0[4] (WCAG) is recommended reading for all web developers, with Guideline 3[5] focusing on the idea of avoiding presentational markup in favor

[4] http://www.w3.org/TR/WCAG10/
[5] http://www.w3.org/TR/WCAG10/#gl-structure-presentation

of semantic markup. As we create projects later in this book, we'll discuss some of these issues more fully.

Standards Compliance

The WCAG isn't the only specification that advocates the use of CSS for the presentational properties of HTML documents. In fact, the latest HTML standards[6] themselves are written with this in mind.

The World Wide Web Consortium[7] (W3C) is the body responsible for publishing recommendations (de facto standards) relating to the Web. Here are some of the W3C Recommendations that relate to using semantic markup and CSS:

HTML 4[8]
: The latest (and last) major revision of the HTML Recommendation marks all non-semantic elements and attributes as **deprecated**.[9] The `font` element, for example, is clearly marked as deprecated in this standard. Under the description of deprecated elements, the Recommendation has this to say:

> In general, authors should use style sheets to achieve stylistic and formatting effects rather than HTML presentational attributes.

XHTML 1.0[10]
: XHTML is a reformulation of HTML 4 as an XML document type. It lets you use HTML tags and attributes while enjoying the benefits of XML features (including the ability to mix tag languages, custom tags, etc.).

  This Recommendation includes the same tags and deprecations as HTML 4.

Web Content Accessibility Guidelines 1.0[11]
: As described in the section called "Accessibility", the WCAG Recommendation strongly recommends using CSS and semantic markup in web design to improve accessibility. I'll let the Recommendation speak for itself:

[6] http://www.w3.org/MarkUp/#recommendations
[7] http://www.w3.org/
[8] http://www.w3.org/TR/html4
[9] A deprecated element or attribute is one that has been tagged for removal from the specification, and which therefore should not be used. For a document to comply strictly with the specification, it should not use any deprecated tags or attributes.
[10] http://www.w3.org/TR/xhtml1/
[11] http://www.w3.org/TR/WCAG10/

Order the print version of this book to get all 500+ pages!

> Misusing markup for a presentation effect (e.g. using a table for layout or a header to change the font size) makes it difficult for users with specialized software to understand the organization of the page or to navigate through it. Furthermore, using presentation markup, rather than structural markup, to convey structure (e.g. constructing what looks like a table of data with an HTML PRE element) makes it difficult to render a page intelligibly to other devices.

Many web developers believe that strict standards compliance is an idealistic goal that is rarely practical. One of the primary goals of this book is to demonstrate that this is not true. Today's browsers provide strong support for CSS and produce more consistent results when they are fed standards-compliant code. While bugs and compatibility issues still exist, they are no more insurmountable than the bugs that face designers who rely on noncompliant code. In fact, once you have valid, standards-compliant code, fixing bugs and compatibility problems can be easier—as you have the starting points of a valid document and style sheet, and just need to find out why the browser display differs—and a lot of help is available on the Web to help you to do that.

Browser Support for CSS

At the time of writing, the browsers employed by the vast majority of web users provide sufficient CSS support to make CSS layouts a viable and sensible choice. The usage of really old browsers—such as Netscape 4—has dwindled to a point where supporting them to the full (i.e. so that these users can access the complete design and functionality of your site) is unnecessary. That said, it's perfectly possible to design sites so that your layout degrades gracefully in older browsers, ensuring that no users are denied access to your content

Designing sites to meet web standards, and constructing them using CSS, should enable you to communicate with more users: they'll be able to access the content whether they're using the latest version of Firefox on a desktop computer, a PDA or phone, an old version of Netscape, or a screen reader. We'll explore some of the ways in which we can optimize site access for various browsers in Chapter 4.

Summary

In this chapter, we explored the primary uses of CSS, and discussed the advantages of designing sites using Cascading Style Sheets. Chapter 3 focuses on the "how"

of CSS: we'll see how rules are included in tags as inline style rules, embedded in pages as embedded style sheets, and loaded from external style sheet files. We'll also investigate in more detail the various selectors and structures of CSS rules, and the units and values you'll use in all rules that require specific measurements.

Order the print version of this book to get all 500+ pages!

3

Digging Below the Surface

This chapter completes our look at the "mechanics" of CSS: the background you need to have in order to work with the technology. It covers six major topics:

- a quick review of the three methods we can use to assign CSS properties to HTML documents
- the use of shorthand properties to group the values for a related set of properties within a single statement
- the workings of the inheritance mechanism in style sheets
- the structure of a style, including variations on the use of selectors to determine with great precision exactly what is affected by a style
- the units and values that can appear in styles to express sizes, locations, and other properties, and how they're used
- CSS comments, which can be used to place human-readable notes in your CSS code

Applying CSS to HTML Documents

In Chapter 1, we discussed three methods for applying style sheet properties to HTML documents. Let's briefly review them here.

inline styles

We can use the `style` attribute, which is available for the vast majority of HTML elements, to assign CSS properties directly to HTML elements.

```
<h1 style="font-family: Helvetica, Arial, sans-serif;
    color: blue;">Welcome</h1>
```

This method is best reserved for times when you want quickly to try out one or more CSS properties to see how they affect an element. You should never use this method in a practical web site, as it avoids almost every advantage that CSS has to offer.

embedded styles

We can use the `style` element in the `head` portion of any HTML document to declare CSS rules that apply to the elements of that page.

```
<style type="text/css">
h1, h2 {
  color: green;
}
h3 {
  color: blue;
}
</style>
```

This form of CSS offers many advantages over inline styles, but is still not as flexible or powerful as external styles (discussed below). I recommend that you reserve embedded styles for use when you're certain that the styles you're creating will be useful only in the current page. Even then, the benefit of separate code offered by external styles can make them a preferable option, but embedded styles can be convenient for quick-and-dirty, single-page work.

external styles

We can use a `<link />` tag in the head portion of any HTML document to apply the CSS rules stored in an external file to the elements of that page.

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


Order the print version of this book to get all 500+ pages!

External styles are the recommended approach to applying CSS to HTML, as this technique offers the full range of performance and productivity advantages that CSS can provide.

Using Shorthand Properties

Most properties take a single item as a value. When you define a property with a collection of related values (e.g. a list of fonts for the `font-family` property), the values are separated from one another by commas, and if any of the values include embedded white space or reserved characters, such as colons, they may need to be enclosed in quotation marks.

In addition, there's a special set of properties called **shorthand properties**, which let you use a single property declaration to assign values to a number of related properties. This sounds more complicated than it is.

The best-known shorthand property is `font`. CSS beginners are usually accustomed to defining font properties one by one:

```
h1 {
  font-weight: bold;
  font-size: 90%;
  line-height: 1.8em;
  font-family: Helvetica, Arial, sans-serif;
}
```

But CSS provides a shorthand property, `font`, that allows this same rule to be defined much more succinctly:

```
h1 {
  font: bold 90%/1.8em Helvetica, Arial, sans-serif;
}
```

You can do the same with properties such as `padding`:

```
h1 {
  padding-top: 10px;
  padding-right: 20px;
  padding-bottom: 10px;
  padding-left: 5px;
}
```

We could replace the above declaration with the following shorthand:

```
h1 {
 padding: 10px 20px 10px 5px;
}
```

The values are specified in a clockwise order, starting at the top of the element: from top, to right, to the bottom, then left.

All shorthand properties are identified in Appendix C.

How Inheritance Works in CSS

Before you can grasp the syntax and behavior of CSS rules, you need a basic understanding of **inheritance**, and how it's used in CSS.

Think of a family tree. Your great-grandfather is at the top of the tree, followed by his children, including his only son (your grandfather). Below your grandfather is your mother and her siblings, and then, beneath her, there's you, your siblings, and your children. Some of your features, such at the color of your hair and eyes, would be inherited from your ancestors—perhaps you have your mother's hair color, but your grandfather's eyes. Other features may not be passed on in this way. Your son may be far taller than anyone else in the family.

Just as everyone in your family fits into your family tree, every element on an HTML page belongs to the document's inheritance tree. The root of that tree is *always* the `html` element.[1] Normally, the `html` element has only two direct descendants in the inheritance tree: `head` and `body`.

Figure 3.1 shows a simple HTML inheritance tree for a small document.

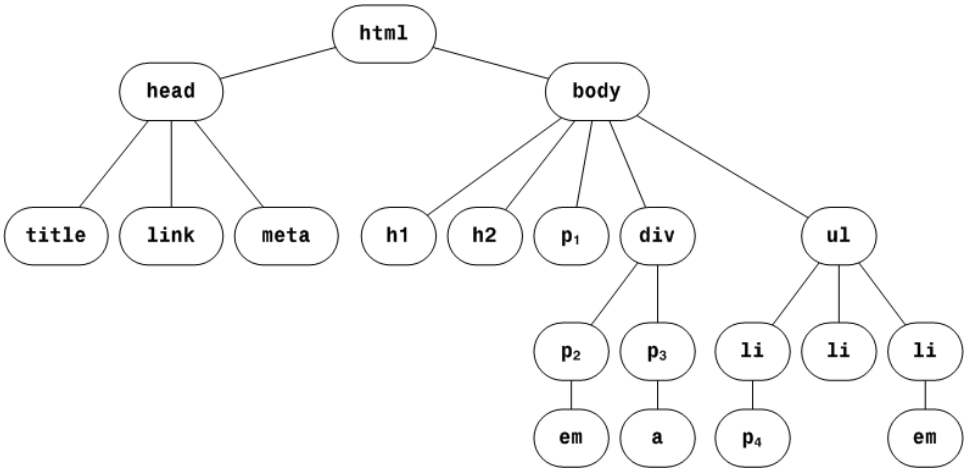
As you can see, the document has in its `head` the standard `title` and `link` elements, the latter of which probably links to an external style sheet. It also includes a `meta` element (most likely to set the document's character set).

The `body` element has five children: an `h1`, an `h2`, a `p` element (labeled p_1 so we can refer to it easily), a `div`, and an unordered list (`ul`) element. The `div` element, in turn, contains two paragraph elements, one of which has an emphasis (`em`) element, while the other contains an anchor (`a`) element. The `ul` element includes three list item (`li`) elements; one of these includes an emphasis (`em`) element, while another contains the paragraph element labeled p_4.

[1] This is even true of documents written to older versions of the HTML standard, in which the `html` element was not required.

Order the print version of this book to get all 500+ pages!

Figure 3.1. A simple HTML inheritance tree



Each element in an HTML document (with the exception of the root `html` element) has a **parent** element. This is the element that directly precedes it in the tree. In Figure 3.1, p_1's parent is the `body` element. Likewise, p_1 is said to be a **child** of the `body` element.

Most elements in an HTML document will be **descendants** of more than one element. For example, in Figure 3.1, the paragraph element p_1 is a descendant of the `body` and `html` elements. Similarly, the paragraph element p_2 is a descendant of the `div`, `body`, and `html` elements. This notion of element hierarchy is important for two reasons:

- The proper use of some of the CSS selectors you'll work with will depend on your understanding of the document hierarchy. There is, for example, an important difference between a descendant selector and a parent-child selector. These are explained in detail in the section called "Selectors and the Structure of CSS Rules", later in this chapter.

- If you don't supply a specific value for an element's property, in many cases, that element will take the value assigned to its parent. Consider the example document shown in Figure 3.1. If the `body` element had a declaration for the `font-family` property and p_1 did not, p_1 would inherit the `body` element's `font-family`. In contrast, setting the `width` property of an element will not directly affect the width of its child elements. `font-family` is an **inherited property**; `width` is not.

The properties that are inherited—and those that are not—are indicated in Appendix C. However, you can set any property to the special value `inherit`, which will cause it to inherit the value assigned to its parent element.

This inheritance issue can become tricky when you're dealing with fairly complex documents. It's particularly important when you're starting with a site that's been defined using the traditional table layout approach, in which style information is embedded in HTML tags. When a style sheet seems not to function properly, you'll often find that the problem lies in one of those embedded styles from which another element is inheriting a value.

Selectors and the Structure of CSS Rules

In Chapter 1 we learned that every CSS style rule consists of two parts: a selector, which defines the type(s) of HTML element(s) to which the style rule applies; and a series of declarations, consisting of properties and values, that define the style.

So far, we've seen only simplistic selectors. Typically, they've contained only one element:

```
h1 {
  font-size: 120%;
  text-transform: capitalize;
}
```

We've encountered one or two instances where a single rule is designed to apply to more than one kind of HTML element:

```
h1, h2, h3 {
  font-size: 120%;
  text-transform: capitalize;
}
```

In this section, we'll take a look at all the different kinds of selectors that are available to you in CSS.

Universal Selector

The **universal selector** matches every element in the document. It has very little practical value by itself, but the universal selector can come in handy in specific

Order the print version of this book to get all 500+ pages!

situations involving, for example, attribute selectors, which I'll explain later in this section.

In this example, all elements in the page are given a text color of red:

```
* {
  color: red;
}
```

Element Type Selector

The **element type selector** is the most common selector. It specifies one HTML element type with no qualifiers. In the absence of other style rules that might apply to the element type provided in the selector, this rule applies to all such elements on the page.

In this example, we specify the text and background color of all hyperlinks in the current document. They will appear as white text on a green background.

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

Class Selector

To apply a style rule to a potentially arbitrary group of elements in a web page, you'll need to define a class in the style sheet, then identify the HTML elements that belong to that class using the `class` attribute.

To define a class in a style sheet, you must precede the class name with a period. No space is permitted between the period and the name of the class.

The following style sheet entry defines a class named `special`.

```
.special {
  font-family: Verdana, Helvetica, Arial, sans-serif;
}
```

Then, we add `class="special"` to the elements that we want to adopt this style.

```
<h1 class="special">A Special Heading</h1>
<p class="special">This is a special paragraph.</p>
```

You can write your class so that it applies only to a particular type of element. In the following example, we create the same `special` class, but this time it applies only to paragraph elements.

```
p.special {
  font-family: Verdana, Helvetica, Arial, sans-serif;
}
```

If you define an element-specific class such as the `p.special` example above, then associate that class (in this case, `special`) with an element of any other type, the style rule simply does not apply to that element.

An HTML element can belong to multiple classes: simply list those classes (separated by spaces) in the `class` attribute:

```
<p class="special exciting">Paragraph! Of! Stuff!</p>
```

ID Selector

An **ID selector** lets you target a single HTML element within a page. Like a class selector, an ID selector must be defined in the style sheet and included explicitly in the HTML tag. Use the # symbol to identify an ID selector in the style sheet,[2] and the `id` attribute to give an element an ID. IDs must be unique within a document; no two HTML elements in a single document should have the same ID.

This style sheet rule defines a rule for an element with the ID `unique`:

```
#unique {
  font-size: 70%;
}
```

The code below uses the HTML `id` attribute to indicate the element that will be affected by the rule above:

```
<h4 id="unique">This will be a very tiny headline</h4>
```

For example, if you had five `<div class="sidebar">` items on your page, but you wanted to style differently the one responsible for displaying your site's search box, you could do so like this:

[2]Optionally, you can confine the ID's use to an element of a specific type by preceding the # with the HTML element's tag name (e.g. `div#searchbox`). But, since you can have only one element with the specific ID within a document, it seems silly to confine it to a specific element type.

Order the print version of this book to get all 500+ pages!

```
div.sidebar {
  border: 1px solid black;
  background-color: yellow;
}
#searchbox {
  background-color: orange;
}
```

The search box would then appear in your HTML as shown here:

```
<div id="searchbox" class="sidebar">
  <!-- HTML for search form -->
</div>
```

Now, since the `div` has `id="searchbox"` and `class="sidebar"` attributes, all the sidebar declarations will be applied to the search box, but it will take its `background-color` from the `#searchbox` rule. The guidelines for cascading overlapping rules (discussed in Chapter 9), in combination with the ID selector, let you avoid having to redefine all the sidebar properties in a special `searchbox` class.

However, you could just as easily define a class and apply it to the exceptional element (the search box, in this example). This approach is more flexible, although perhaps not as efficient in terms of code space. For example, imagine you've identified a class or other rule that applies to all level-three headings except one, and you've used an ID selector for the exception. What do you do when a redesign or content change requires one more such exception? The ID selector solution breaks down immediately in that situation.

Pseudo-element Selector

This and all the remaining selectors in this section require a browser that supports the CSS 2 specification, such as Firefox, Safari, Opera, or Internet Explorer 7. Some features, such as the `:hover` pseudo-class, are supported by some older browsers, but their implementations are not complete.

Pseudo-element selectors and pseudo-class selectors are unique among the CSS selectors in that they have no equivalent HTML tag or attribute. That's why they use the prefix "pseudo" (meaning "false").

So far, the CSS specification has defined only three pseudo-elements: `first-letter`, `first-line`, and `first-child`. While the first two of these phrases mean something to us humans, it's ultimately up to each browser to interpret

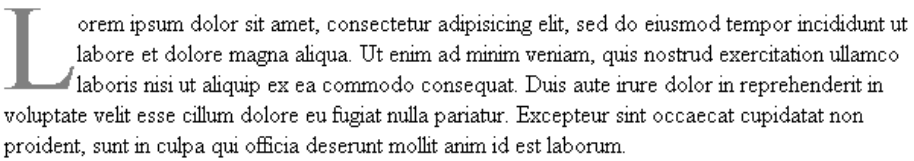
them when rendering HTML pages that use these pseudo-elements. For example, does `first-line` mean "first sentence," or does it mean the first physical line that's displayed—a value that changes as the user resizes the browser? The `first-child` pseudo-element, on the other hand, is not browser-dependent. It refers to the first descendant of the element to which it is applied, in accordance with the HTML document hierarchy described in the section called "How Inheritance Works in CSS".

To define a pseudo-element selector for a style rule, precede the pseudo-element name with a colon. Here's an example:

```
p:first-letter {
  font-face: serif;
  font-size: 500%;
  float: left;
  color: gray;
}
```

This creates a drop-caps effect for the first letter in every paragraph on the page, as shown in Figure 3.2. The first letter in each paragraph will be five times larger than the usual type used in paragraphs. The `float` style property, which we discuss in Chapter 8, ensures the remaining text in the paragraph wraps around the enlarged drop-cap correctly.

Figure 3.2. Creating a drop-caps effect using the `first-letter` pseudo-element



Pseudo-class Selector

A **pseudo-class selector** is exactly like the pseudo-element selector, with one exception. A pseudo-class selector applies to a whole element, but only under certain conditions.

The current release of CSS 2 defines the following pseudo-classes:

Order the print version of this book to get all 500+ pages!

- `hover`
- `active`
- `focus`
- `link`
- `visited`
- `lang`

A style sheet, then, can define style rules for these pseudo-classes as shown in the example below. You may remember that we've already seen a rule that uses the `hover` pseudo-class.

```
a:hover {
  color: green;
}
```

All anchor tags will change color when the user mouses over them. As you can see, this means the pseudo-class selector comes into play only when the user interacts with the affected element.

The `lang` pseudo-class[3] refers to the setting of the `lang` attribute in an HTML element. For example, you could use the `lang` attribute shown below to define a paragraph in a document as being written in German:

```
<p lang="de">Deutsche Grammophon</p>
```

If you wanted, for example, to change the font family associated with all elements in the document that were written in German, you could write a style rule like this:

```
:lang(de) {
  font-family: spezialitat;
}
```

`lang` vs `language`

Be careful not to confuse this `lang` attribute with the deprecated `language` attribute that used to be used to set the scripting language used in pages.

[3]Be aware that browser support for the `lang` pseudo-class is still very scarce. It's covered here mainly for the sake of completeness.

Descendant Selector

As we've discussed, all HTML elements (except the `html` element) are descendants of at least one other HTML element. To apply a CSS style rule to an element only when it's a descendant of some other kind of element, we can use a **descendant selector**.

A descendant selector, such as the one shown in the following style rule, restricts the applicability of the rule to elements that are descendants of other elements. The scope of the descendant selector is determined by reading the rule from right to left. Spaces separate the element types.

```
li em {
  color: green;
}
```

The style rule identifies that a color of green will be applied to any text contained in an `em`, or emphasis, element *only* when the emphasized text is a descendant of a list item.

In the fragment below, the first `em` element will be displayed in green characters; the second will not, as it doesn't appear within a list item.

```
<ul>
  <li>Item one</li>
  <li>Item <em>two</em></li>
</ul>
<p>An <em>italicized</em> word.</p>
```

It's important to note that the descendant relationship need not be an immediate parent-child connection. Take this markup, for example:

```
<div class="sidebar">
  <p>If you have any questions, <a href="contact.html">please call
      our office during business hours</a>.</p>
</div>
```

The following style rule would apply to the anchor element even though it focuses explicitly on `a` elements that are descendants of `div` elements. This is because, in this case, the `a` element is the child of a paragraph that's contained in a `div` element.

Order the print version of this book to get all 500+ pages!

```
div a {
  font-style: italic;
}
```

Parent-child Selector

A **parent-child selector** causes a style rule to apply to element patterns that match a specific sequence of parent and child elements. It is a special case of the descendant selector that we discussed above. The key difference between the two is that the pair of elements in a parent-child selector must be related directly to one another in a strict inheritance sequence.

A parent-child relationship is specified in a selector with the "greater than" sign (`>`).

Below is an example of a parent-child relationship.

```
body > p {
  font-weight: bold;
}
```

In the example below, this rule will only affect `para2`, as `para1` and `para3` are not direct descendants of the `body` element.

```
<body>
  <div class="sidebar">
    <p id="para1">This is the sidebar.</p>
  </div>
  <p id="para2">Welcome to the web site! Here's a list:</p>
  <ul>
    <li>
      <p id="para3">This is the first paragraph in the list. It's
          also the last.</p>
    </li>
  </ul>
</body>
```

As of this writing, Internet Explorer for Windows (up to and including version 6) distinguishes itself by being the only major browser that does not support parent-child selectors. Because of this, careful use of descendant selectors is far more common, and the parent-child selector is often abused to specifically create styles that do not apply to Internet Explorer for Windows.

Adjacent Selector

Adjacency is not related to inheritance. Adjacency refers to the sequence in which elements appear in an HTML document. As it happens, adjacent elements are always siblings, but it's their placement in the document, rather than their inheritance relationship, that is the focus of this selector. This point is demonstrated in the HTML fragment below:

```
<h1>This is important stuff!</h1>
<h2>First important item</h2>
<h2>Second important item</h2>
```

The first `h2` heading is *adjacent* to the `h1` heading, but the second `h2` heading is not adjacent to the `h1` heading.

The adjacent selector uses the `+` sign as its connector, as shown here:

```
h1 + h2 {
  margin-top: 11px;
}
```

This style rule would put 11 extra pixels of space between the bottom of an `h1` heading and an `h2` heading that followed it immediately. It's important to recognize that an `h2` heading that follows a paragraph under an `h1` heading would not be affected.

As of this writing, Internet Explorer for Windows (up to and including version 6) remains the only major browser that does not support adjacent selectors, although support is planned for Internet Explorer version 7. Because of this, the adjacent selector has not yet found widespread use in practical web design.

Attribute Selectors

The group of selectors I'm lumping together as **attribute selectors** are among the most interesting of all the CSS selectors, because they almost feel like programming techniques. Each attribute selector declares that the rule with which it is associated is applied only to elements that have a specific attribute defined, or have that attribute defined with a specific value.

There are four levels of attribute matching:

`[attribute]`	matches if the attribute *`attribute`* is defined at all for the element(s)

Order the print version of this book to get all 500+ pages!

`[attribute="value"]`	matches only if the attribute has a value of *value*
`[attribute~="value"]`	matches only if the attribute is defined with a space-separated list of values, one of which exactly matches *value*
`[attribute|="value"]`	matches only if the attribute is defined with a hyphen-separated list of "words," and the first of these words begins with *value*

You might, for example, want to apply style properties to all single-line text input boxes (`<input type="text" />`) in your document. Perhaps you want to set their text and background colors to white and black, respectively. This style rule would create that effect:

```
input[type="text"] {
  color: white;
  background-color: black;
}
```

The third variation of the attribute selector described above searches the values assigned to an attribute, to see whether it contains the word you've specified (i.e. a value in a space-separated list).

For example, during the development of a web site, various graphic designers may have inserted some `img` elements with temporary placeholder `alt` attributes, with the idea of returning to them later to finish them. You could call attention to the existence of such tags with a style rule like this:

```
img[alt~="placeholder"] {
  border: 8px solid red;
}
```

This selector will find all `img` elements whose `alt` attributes contain the word "placeholder," and will put an eight-pixel red border around them. That ought to be hard to miss!

The fourth variation really is useful only when you're dealing with the `lang` attribute. Typically, the `lang` attribute takes on a value such as `en` or `de`. However, it can also be used to define the regional dialect of the language being used: `en-us` for American English, `en-uk` for British English, etc. This is when the `[attribute|="value"]` selector comes into its own. It enables you to isolate the first portion of the `lang` attribute, where the language that's being used is defined. The other portions of the hyphen-separated value are ignored.

As you've probably come to expect by now, attribute selectors are not supported by Internet Explorer for Windows versions 6 and earlier. As with other advanced selector types, this has prevented the widespread adoption of attribute selectors, despite their obvious usefulness.

Selector Grouping

To apply a style rule to elements of several different types in an HTML document, we use selector grouping, separating with a comma the element types to which the rule is to be applied.

Here's a simple example of this type of selector:

```
h1, h2, h3 {
  font-family: Helvetica, Arial, sans-serif;
  color: green;
}
```

The elements in the selector list need not be of the same type or even the same level of specificity. For example, the following style rule is perfectly legal. It applies a specific style to level-two headings (`h2`) and to paragraphs whose class is defined as `special`:

```
h2, p.special {
  font-size: 22px;
}
```

You may include a space between the comma-separated items, though this is not necessary.

Expression Measurements

Most of the values we define in a CSS rule include measurements. These measurements tell the rule how tall or wide something is to be, so it follows that you'll most commonly use measurements when working with fonts, spacing, and positioning.

There are two types of measurements: absolute and relative. An absolute measurement (e.g. setting a `font-size` to `18px`, or 18 pixels) tells the browser to render the affected content 18 pixels tall.[4] Technically speaking, it tells the browser to

[4]Again, if I wanted to be terribly precise, I would say that a pixel is actually a relative measurement, because its meaning is relative to the display medium on which the page is produced. But, in this

Order the print version of this book to get all 500+ pages!

use the specified font and scale its character height so that the font's overall height is 18 pixels. Chapter 8 includes an explanation of font height and width.

Relative measurements, on the other hand, instruct the browser to scale a value by some percentage or multiple, relative to the size of the object before the scaling takes place. The example below defines a style rule in which all fonts in paragraphs on the page should be scaled to 150% of the size they would have been without this style:

```
p {
  font-size: 150%;
}
```

If you knew that, in the absence of such an instruction, the text of all paragraphs on the page displays at a size of 12 pixels, you could also accomplish the same thing this way:

```
p {
  font-size: 18px;
}
```

Generally, you should use the relative sizing values whenever you can. This technique works better than absolute sizing when the user has set preferences for font sizes, and in situations in which multiple style sheets could be applied. It's also more accessible, as visually impaired users can more easily increase the font size on the page by configuring their browsers' preferences.

All **length values** (the term used by the CSS specification to describe any size measurement, whether horizontal or vertical) consist of an optional sign (`+` or `-`), followed by a number (which may include a decimal point), followed by a unit of measurement. No spaces are permitted between the number and the unit of measurement.

context, "relative" means "relative to some other value in the style rule or in the HTML," and in that sense, pixels are absolute.

Absolute Values

Table 3.1. Absolute values supported in style sheets

Style Abbreviation	Style Meaning	Explanation
in	inch	Imperial unit of measure; 2.54 centimeters
cm	centimeter	
mm	millimeter	
pt	point	1/72 inch
pc	pica	12 points, or one-sixth of an inch
px	pixel	One dot on the screen

Table 3.1 shows the absolute values that are supported in CSS style sheets, and where they're not obvious, the values' meanings.

When a length of zero is used, no unit of measurement is needed. `0px` is the same as `0`. It doesn't make sense to give a unit of measurement when the length is zero units, because zero is the same distance in any unit of measurement.

Whenever you need to supply an absolute measurement for the size or position of an element in a style sheet rule, you can use any of the above abbreviations interchangeably. Each of the following rules should produce precisely the same result:

```
font-size: 1in;
font-size: 2.54cm;
font-size: 25.4mm;
font-size: 72pt;
font-size: 6pc;
```

Pixels pose an entirely different set of issues. A pixel is one point on a screen that can be on or off, displaying any color that is needed. If you set your monitor's display to a resolution of 800 pixels by 600 pixels, a pixel corresponds to 1/600 of the screen height. On a 15-inch display, the height is about 10.5 inches and

Order the print version of this book to get all 500+ pages!

the width is a little more than 13 inches.[5] A 12-pixel font display on that monitor would turn out to be about 1/50 of the 10.5-inch height of the display, or just a little more than one-fifth of an inch.

Many designers set their font sizes using pixels in the belief that this prevents site users from increasing the font size using their browser settings, because Internet Explorer does not allow the resizing of text set in pixels. However, most other browsers do allow the user to resize text set in pixels. A common issue arises with sites whose designers haven't realized that fonts set using pixels can be resized in other browsers: often, the text will appear to expand out of fixed-size boxes. From the point of view of accessibility, if users need a larger font size and have increased the text size in their browsers accordingly, we should support this choice regardless of which browser they're using; thus, we should avoid setting text heights using pixels. Creating designs that work well even if users have increased the text size in their browsers is part of the process of designing for the Web. The use of pixels to size text should be avoided.

Relative Values

Because of the problems posed by the use of any absolute value, the most flexible way to approach measurements for style rules is to use relative units of measurement. Principally, these units are `em` and percentage, although some people prefer to use the more obscure `ex` measurement. The `em` measurement is so named because it refers to the width of an uppercase "M" in the given font, but in practice, it's equal to the `font-size` of the current font. The `ex` measurement is based on the height of the lowercase "x" character in a font (more commonly known as the **x-height** of the font) and is far less common than the `em`.

Both the `em` and the percentage generate font sizes based on the inherited or default size of the font for the object to which they're applied. In addition, `em`s and percentages are 1:100 equivalent. A size of `1em` is identical to a size of `100%`.

This description begs the question, "What's the default or inherited font size for a particular HTML element?" The answer is: it depends.

Prior to the emergence of Opera 5 for Windows, browsers set the default values for all fonts as part of their startup processes. Users had no control. The browsers

[5]High school math would lead you to predict a nine- by 12-inch screen, but unfortunately, 15-inch monitors don't normally have a full 15 inches of diagonal screen space. Perhaps computer manufacturers don't study Pythagoras.

defined a default, and web designers overrode the defaults willy-nilly, as they saw fit. The user took what was presented.

Then, along came the idea of user choice—a development that, not surprisingly, was facilitated by the emergence of CSS. Essentially, the developers of the Opera browser created a local style sheet that users could modify and set their own defaults to use. The Opera developers also defined a nice graphical user interface through which users could set preferences for these styles.

This was great for users, but web designers found themselves in a quandary. If, for example, you assumed that browsers were going to default body text to a 12-point font size[6] (which was the de facto standard before the user-controlled preferences era), you could set a style to apply a `1.25em` scaling to the text and get a 15-point font size for the text in question. It was nice and predictable.

But now, a `1.25em` scaling applied to a font tells the browser to increase the size of the font to 1.25 times (or 125% of) its default size. If the user has set up his or her browser to show standard text at a height of 16 points, your `1.25em` transformation brings the size up to 20 points.

When you stop to think about it, though, that's probably just fine. The user who chooses a larger base font size probably needs to see bigger type. If you want type that would otherwise be at 12 points to display at 14 for some good reason, then it's not unreasonable to expect that this new user will benefit in the same way from seeing the font used in this particular situation increase from his or her standard 16 points to 20.[7]

Most of the time, there's not really a reason to muck around with the user's settings for font sizes, so changing them arbitrarily isn't a good idea. Before you apply this kind of transformation to a segment of text in your web design, ask yourself if it's really necessary. My bet is that, nine times out of ten, you'll find it's not.

I would be remiss if I didn't point out that some pitfalls are inherent in the use of relative font sizes. Under some circumstances, relative font values can combine and multiply, producing bizarre results indeed.

[6]Just in case you were wondering, pixel sizes and point sizes are not equivalent, and the ratio between the two varies between browsers and operating systems. For example, the 12-point default font size used by most Windows browsers was rendered at 16 pixels on that platform. `12pt` is equivalent to `16px` on Windows browsers.

[7]If that's not the case, you probably want to rethink your reason for boosting the font size in the first place.

Order the print version of this book to get all 500+ pages!

For example, let's say that you define style rules so that all text that's bold is displayed at `1.5em` and all italic text is displayed at `1.5em`, as shown below.

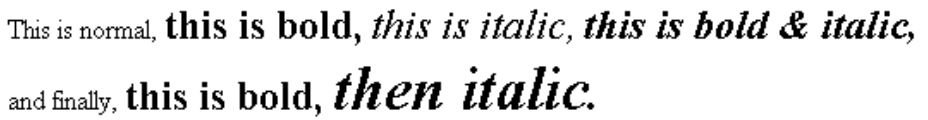
```
.bold {
  font-weight: bold;
  font-size: 1.5em;
}
.italic {
  font-style: italic;
  font-size: 1.5em;
}
```

In your document, these styles are used together in a number of different ways, as shown in this markup:

```
<p>This is normal, <span class="bold">this is bold,</span>
   <span class="ital">this is italic,</span>
   <span class="bold ital">this is bold and italic,</span> and
   finally, <span class="bold">this is bold,
   <span class="ital">then italic</span></span>.</p>
```

When you nest[8] these styles, the resulting text will display at `2.25em` (`1.5em` × `1.5em`). This problem arises with child elements, which inherit from their parent container elements the computed values for measured properties, not the relative values. This is relatively easy to avoid, but if you overlook it, the results can be quite startling, as Figure 3.3 illustrates.

Figure 3.3. Relative measurements gone haywire



CSS Comments

You're probably already familiar with the concept of **comments** in HTML:

```
<!-- this is an HTML comment -->
```

[8]Nesting is the process of putting one element inside another. For example, we say that a `span` inside another `span` is nested.

Comments allow you to include explanations and reminders within your code. These are ignored entirely by the browser, and typically are included solely for the developer's convenience. If you've ever had to make changes to code that hasn't been touched in a few months, I'm sure you can appreciate the value of a few well-placed comments that remind you of how it all works.

CSS has its own syntax for comments. In HTML, a comment begins with `<!--` and ends with `-->`. In CSS, a comment begins with `/*` and ends with `*/`:

```
<style type="text/css">
  /* This rule makes all text red by default. We include
     paragraphs and table cells for older browsers that don't
     inherit properly. */
  body, p, td, th {
    color: red;
  }
</style>
```

If you know much JavaScript, you'll recognize this syntax, which can be used to create multiline comments in that language as well. However, unlike JavaScript, CSS does not support the single-line double-slash (`//`) comment style.

Summary

This chapter ended our overview of CSS technology with a tour of some of the syntactic and structural rules of CSS styles. Along the way, it explained the basic ideas involved in HTML document inheritance.

In Chapter 4, we'll see how you can check your pages to see if they meet the W3C Recommendations. Passing such a check will help you ensure that your pages will display as expected not only in current browsers, but in all future browsers as well. We'll also learn a few tricks to get your pages to display in a usable way in older browsers.

Order the print version of this book to get all 500+ pages!

8

Simple CSS Layout

We now have some sound theory under our belts. The rest of this book will concentrate on how you can put CSS into practice when developing your own sites. Along the way, we'll be learning how to lay out pages using CSS—moving from simple layouts to more complex ones—and how you can combine some of the concepts you've already read about to create great-looking sites.

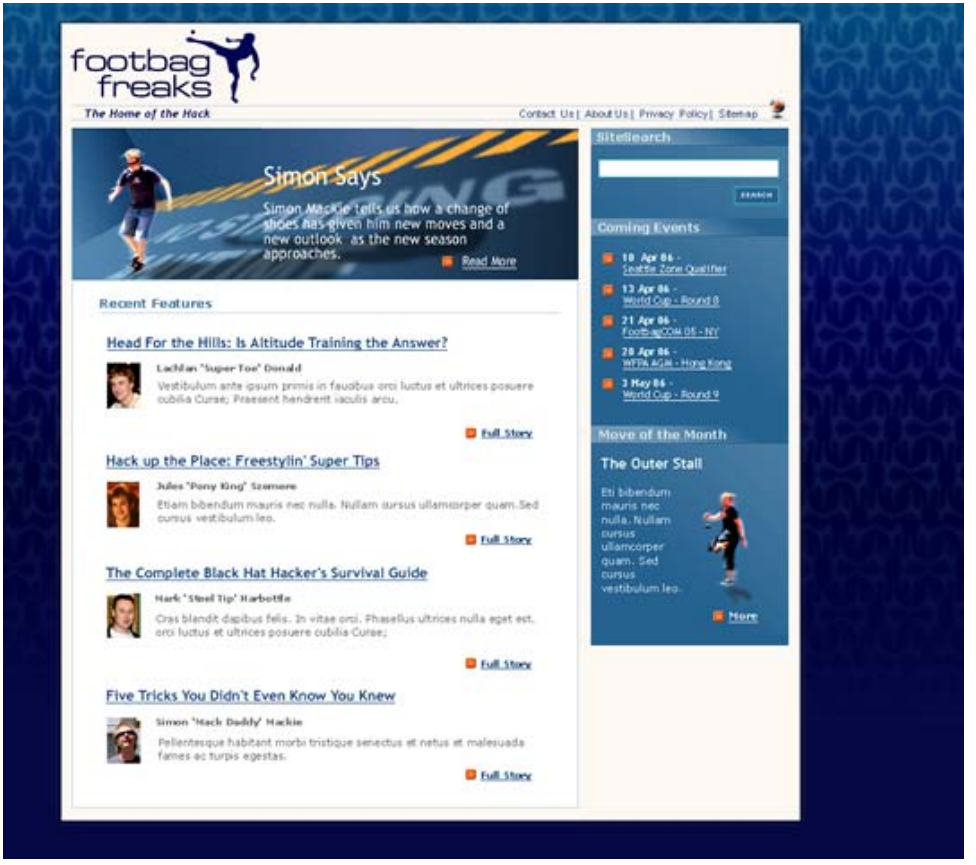
This chapter will start with the creation of a simple two-column layout. Along the way, we'll discover how to use absolute and relative positioning, and see how margins, padding, and borders work together. Then, we'll get an understanding of how all these tools can be used together in practice by creating a two-column layout that uses many of the techniques we have discussed already in this book.

While the layout we'll create in this chapter is a relatively simple one, it's a structure that's used by many web sites; the layout we'll develop here could easily form the basis for a production site.

The Layout

Many web site designs start life as mock-ups in a graphics program. Our first example site is no exception: we have an example layout or "design comp" created in Fireworks as a starting point.

Figure 8.1. Creating the layout as an image file

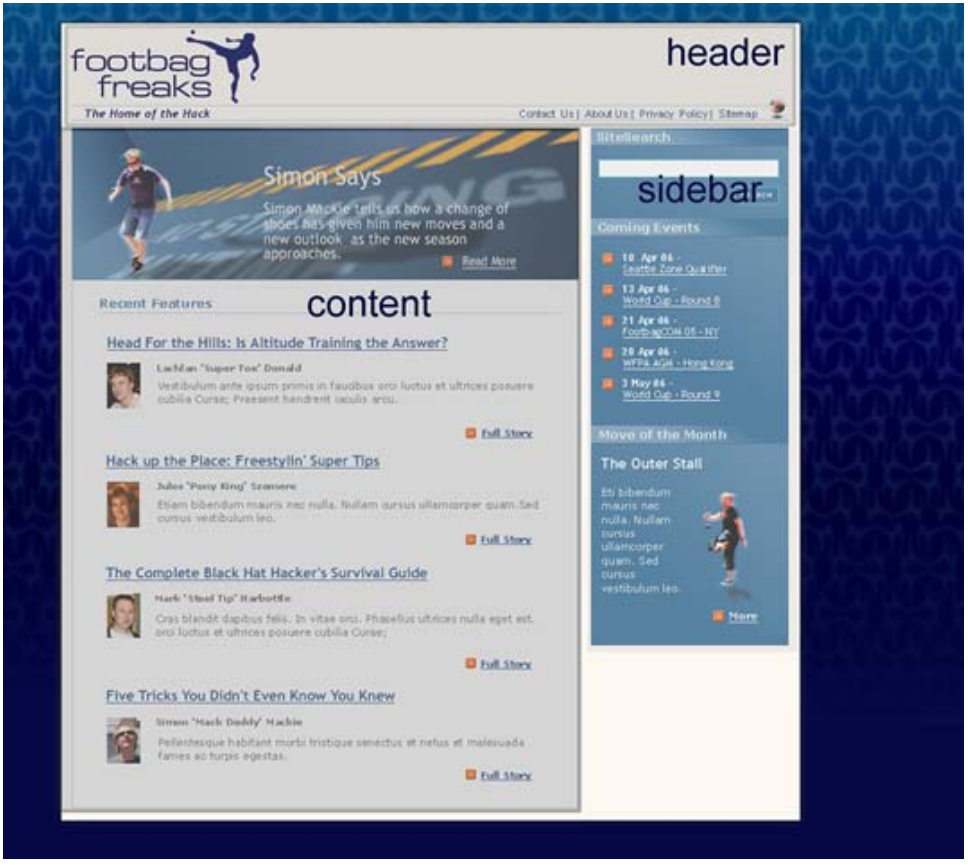


Starting out with a visual like this enables us to think about the way we're going to build the site *before* we start to write any XHTML or CSS. It gives us the opportunity to decide how best to approach this particular layout before we code a single line.

This layout divides the page into three main sections: a header, which contains the site logo and some main navigation; a main content area comprising a large image above a list of news stories; and a sidebar, which presents some additional items.

Order the print version of this book to get all 500+ pages!

Figure 8.2. Marking the main sections on the layout



This layout could be described as a two-column layout with a header area. Being able to visualize a design as being a combination of its main sections eases the process of deciding how to approach the page layout.

Creating the Document

Having decided what the basic components of our page will be, we can start work. The first thing we'll do is create an XHTML document that contains all of the text elements we can see in our layout image, marked up using the correct XHTML elements.

Working this way might seem a little strange at first, particularly if you have been used to working in a visual environment, such as Dreamweaver, and simply concentrating on how the design looks. However, one of the advantages of using CSS for layout is that we're able to separate the structure of the page from its appearance. This allows us to concentrate on building a good solid document as the basis of our site, before adding the design using CSS.

We start out with the basic requirements for an XHTML Strict document. As we're going to use CSS for all of the presentational information on this site, there's no reason not to use a Strict DOCTYPE. The Transitional DOCTYPEs (for both XHTML and HTML 4.01) allow you to use attributes and elements that are now deprecated in the W3C Recommendations. The deprecated elements and attributes are mainly used for presentation, and as we're going to use CSS—not XHTML—for presentation, we won't need to use these anyway.

File: **index.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Footbag Freaks</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
  </head>
  <body>
  </body>
</html>
```

Declaring the Character Set

In our pages, we've used the `meta` element with the `http-equiv="Content-Type"` attribute to declare our document's character set. This makes it easy for browsers (and the W3C validator) to determine which character set is being used in the document. If this information was missing, a browser could misinterpret the characters in your page, which could see your pages rendered as unintelligible garbage.

All of the examples in this book use ISO-8859-1 encoding, which is the default for most popular text editors and programs such as Dreamweaver. If you're dealing with a different character set, such as Unicode, you'll need to change the `meta` elements accordingly.

Order the print version of this book to get all 500+ pages!

The Header

Let's start to add the content of this page to our document. As we do so, we'll split it up into the various sections identified above, containing each page section between `<div>` and `</div>` tags. We'll give each `div` an `id` to identify that section; we'll use these `id`s to address each section and style it using CSS.

After the `<body>` tag, add the following markup:

File: **index.html (excerpt)**

```
<div id="header">
  <p>The Home of the Hack</p>
  <ul>
    <li><a href="">Contact Us</a></li>
    <li><a href="">About Us</a></li>
    <li><a href="">Privacy Policy</a></li>
    <li><a href="">Sitemap</a></li>
  </ul>
</div> <!-- header -->
```

We won't worry about any image elements at this point, because there are numerous ways in which we can add images to the page using CSS; we'll make the decision as to the best way to add each image as we create our CSS. Thus, the header area simply contains the tag line, "The Home of the Hack," and a list that includes the main navigation links.

The Main Content Section

The main content section comes next, contained in a `div` with an `id` of `content`:

File: **index.html (excerpt)**

```
<div id="content">
  <h2>Simon Says</h2>
  <p>Simon Mackie tells us how a change of shoes has given him new
      moves and a new outlook as the new season approaches.</p>
  <p><a href="">Read More</a></p>
  <h2>Recent Features</h2>
  <ul>
    <li>
      <h3>Head for the Hills: Is Altitude Training the
          Answer?</h3>
      <p>Lachlan 'Super Toe' Donald</p>
      <p>Vestibulum ante ipsum primis in faucibus orci luctus et
```

```
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>Hack up the Place: Freestylin' Super Tips</h3>
        <p>Jules 'Pony King' Szemere</p>
        <p>Vestibulum ante ipsum primis in faucibus orci luctus et
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>The Complete Black Hat Hacker's Survival Guide</h3>
        <p>Mark 'Steel Tip' Harbottle</p>
        <p>Vestibulum ante ipsum primis in faucibus orci luctus et
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>Five Tricks You Didn't Even Know You Knew</h3>
        <p>Simon 'Mack Daddy' Mackie</p>
        <p>Vestibulum ante ipsum primis in faucibus orci luctus et
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
    </ul>
</div> <!-- content -->
```

This area will contain the large image with a text overlay that highlights a feature story. Four news items will be listed below this.

The Sidebar

Finally, let's add the sidebar, which contains a search box and some important dates:

File: **index.html (excerpt)**

```
<div id="sidebar">
  <h3>Site Search</h3>
  <form method="post" action="" id="searchform">
    <div>
      <label for="keywords">Keywords</label>:
```

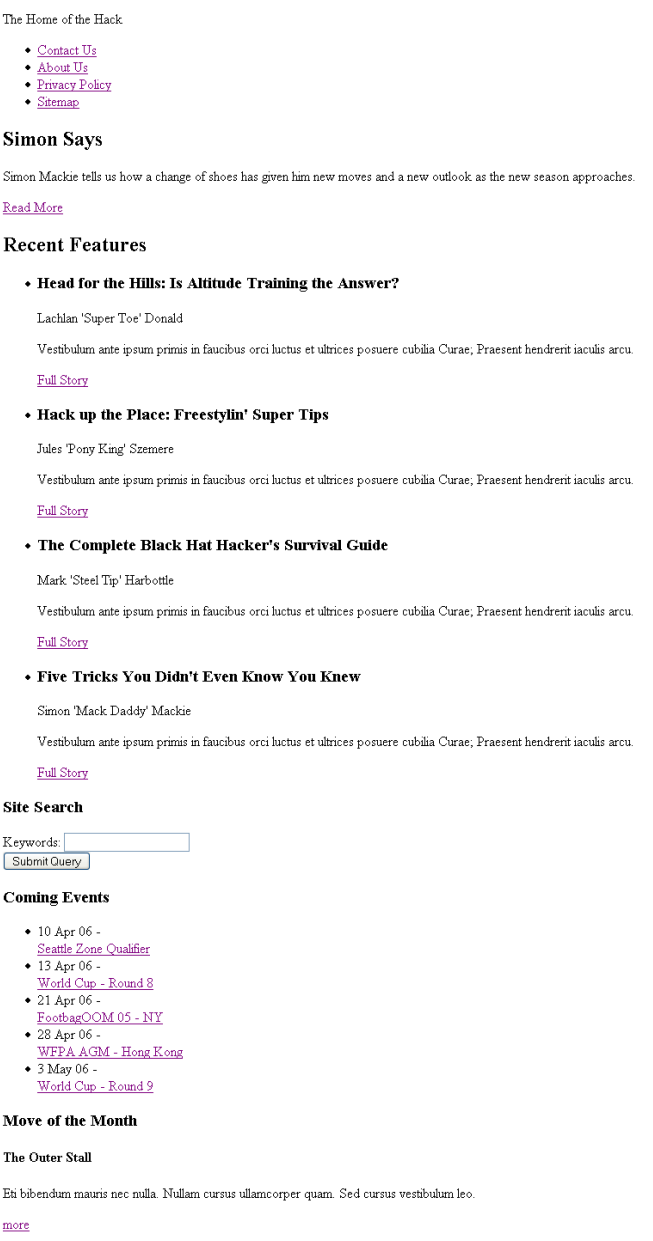

Order the print version of this book to get all 500+ pages!

```
      <input type="text" name="keywords" id="keywords" />
    </div>
    <div>
      <input type="submit" name="btnSearch" id="btnSearch" />
    </div>
  </form>
  <h3>Coming Events</h3>
  <ul>
    <li>10 Apr 06 -<br /><a href="">Seattle Zone
        Qualifier</a></li>
    <li>13 Apr 06 -<br /><a href="">World Cup - Round 8</a></li>
    <li>21 Apr 06 -<br /><a href="">FootbagOOM 05 - NY</a></li>
    <li>28 Apr 06 -<br /><a href="">WFPA AGM - Hong Kong</a></li>
    <li>3 May 06 -<br /><a href="">World Cup - Round 9</a></li>
  </ul>
  <h3>Move of the Month</h3>
  <h4>The Outer Stall</h4>
  <p>Eti bibendum mauris nec nulla. Nullam cursus ullamcorper
      quam. Sed cursus vestibulum leo.</p>
  <p><a href="">more</a></p>
</div> <!-- sidebar -->
```

This completes our markup for the homepage of the site. Save your page and view it in your browser. The content of your document will display using the default styles for the elements that we've used, as Figure 8.3 illustrates. It won't be pretty, but it should be easily readable!

Our last job before we start to add the CSS that will create the design we see in the example graphic is to validate our markup. By validating the document at this point, we'll know that we're adding CSS to a valid document: we won't come up against problems caused by existing invalid markup.

Figure 8.3. Displaying the page after the content is added



Order the print version of this book to get all 500+ pages!

Positioning the Page Elements

We can now begin to create our style sheet. But, before we do, we need to take a moment to understand some basic concepts that come into play when creating layouts such as this (and many others): the `display` property, the concept of positioning, and the CSS Box Model technique.

The `display` Property

Before we can move on to look at CSS positioning issues, we should take a quick look at the `display` property, as it can have a significant impact on page layout.

The `display` property determines how a browser displays an element—whether it treats it as a block, an inline text fragment, or something else. Although it can be assigned any of 17 legal values, browser support realities confine the list to six, only four of which are really important. For a full reference to `display` see Appendix C.

The six possible values for the `display` property are:

- `block`
- `inline`
- `list-item`
- `none`
- `table-footer-group`
- `table-header-group`

The default value varies from element to element. Block elements such as `p`, `h1`, and `div` default to `block`, while inline elements (those that would normally occur within a section of text), such as `strong`, `code`, and `span`, default to `inline`. List items default to `list-item`. Assigning non-default settings to elements can produce interesting and useful effects. Later in this book, we'll see how we can use `display: inline` to cause a list to display horizontally.

If you supply a value of `none`, the element to which it applies will not display, and the space it would normally occupy will be collapsed. This differentiates the `display: none` declaration from the `visibility: hidden` declaration, which is

commonly used to hide an element but preserve the space it would occupy if it were visible.

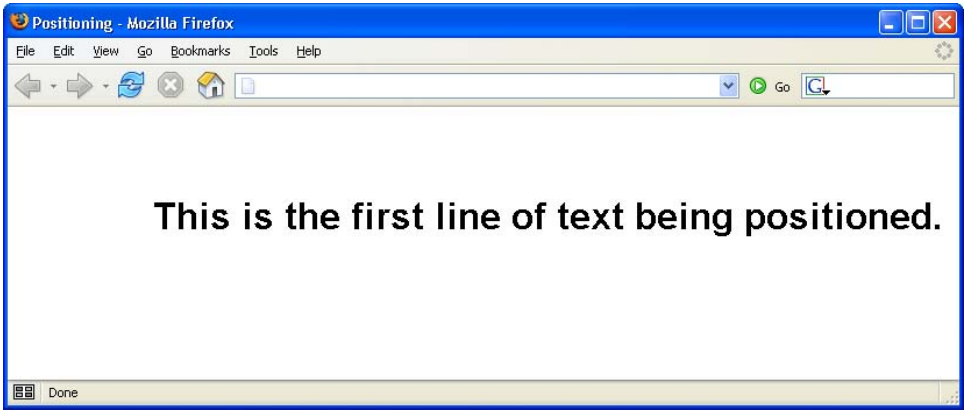
Absolute, Relative, and Positioning Contexts

The CSS `position` property takes on a single, constant value that determines how the block is positioned on the page. The two most frequently used values are `absolute` and `relative`. Another value, `static`, is the default value for this property; the fourth value, `fixed`, is not supported by Internet Explorer 6.

Positioning in CSS can be confusing because the points that are referenced to guide a block's placement on the page change in accordance with the **positioning context** of the block. There's no universal set of coordinates to guide placement, even when you're using the `absolute` positioning value. Each time a block is positioned on the page with a `position` setting other than `static`, it creates for its descendants a new positioning context in which the upper left corner of its content area has the coordinates (0,0). So, if you use CSS to position an element within that block, its position will be calculated relative to that new coordinate system—its "positioning context."

The best way to understand this concept is to look at a few simple, interrelated examples. Let's start with a blank page. In this context, the upper left corner of the viewport—the viewable area of the browser window—is where the initial (0,0) coordinates are located. Let's place a simple piece of text in a `div`, as shown in Figure 8.4.

Figure 8.4. The first line of text



Order the print version of this book to get all 500+ pages!

Here's the HTML fragment that produces the result shown above. The CSS properties `top` and `left` are used to position the `div` on the page, locating it 75 pixels from the top of the page, and indenting it from the left of the page by 125 pixels:

File: **positioning.html (excerpt)**

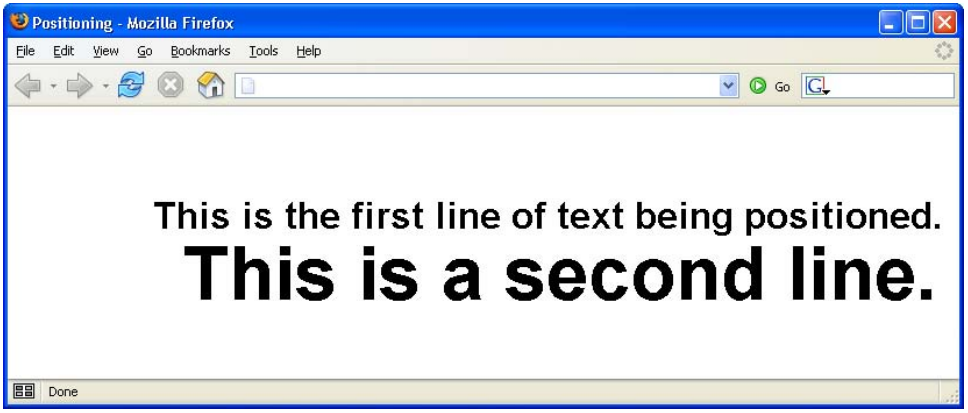
```
<div style="position: absolute; left: 125px; top: 75px;"
    class="big">
  This is the first line of text being positioned.
</div>
```

Now, put a second `div` inside the first one, as shown here:

File: **positioning.html (excerpt)**

```
<div style="position: absolute; left:125px; top: 75px;"
    class="big">
  This is the first line of text being positioned.
  <div style="position: absolute; left: 25px; top: 30px;"
      class="big">
    This is a second line.
  </div>
</div>
```

Figure 8.5. An element positioned inside a positioned block



The result is shown in Figure 8.5. Notice that the second line of text is indented 25 pixels from the left of the first line of text, because that first line sets the positioning context for the second: it's the parent element of the second line. Both lines are positioned absolutely; however, the first line is positioned from the top

and left of the viewport, and the second line is positioned absolutely from the top and left of the first. Notice, too, that its font size is huge. Why? Take a look at the style rule for the `big` class, and you'll understand:

File: **positioning.html (excerpt)**

```
.big {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 2em;
  font-weight: bold;
}
```

As the second `div` is a child of the first, its font size is calculated relative to that of the first `div`. The style rule defines the font as being of size two ems, which instructs the browser to render the text at twice the size it would otherwise appear. When that two em rule is applied to the first line, its size is doubled. But when it is applied to the second line, the font size of the first line is doubled to calculate that of the second.

We can correct this using an absolute font size constant:

File: **positioning.html (excerpt)**

```
.big {
  font-family: Helvetica, Arial, sans-serif;
  font-size: large;
  font-weight: bold;
}
```

The two `div`s should now share the same font size.

The page now has two `div` elements, one nested inside the other. Both use absolute positioning. Now, let's add a third element—this time, a `span` element that will be contained in the second `div`. Using relative positioning, the HTML looks like this:

File: **positioning.html (excerpt)**

```
<div style="position: absolute; left: 125px; top: 75px;"
    class="big">
  This is the first line of text being positioned.
  <div style="position: absolute; left: 25px; top: 30px;">
    This is <span
        style="position: relative; left: 10px; top: 30px;">an
        example of</span> a second line.
 </div>
</div>
```

 Order the print version of this book to get all 500+ pages!

The result of this markup can be seen below. Notice that the words "an example of," which are contained in the `span`, appear below and slightly to the right of their original position. *Relative positioning is always based on the positioned element's original position on the page*. In other words, the positioning context of an element that uses relative positioning is provided by its default position. In this example, the `span` is positioned as shown in Figure 8.6. It appears below and to the right of where it would normally be if no positioning was applied—a case that's illustrated in Figure 8.7.

Figure 8.6. Example of relative positioning

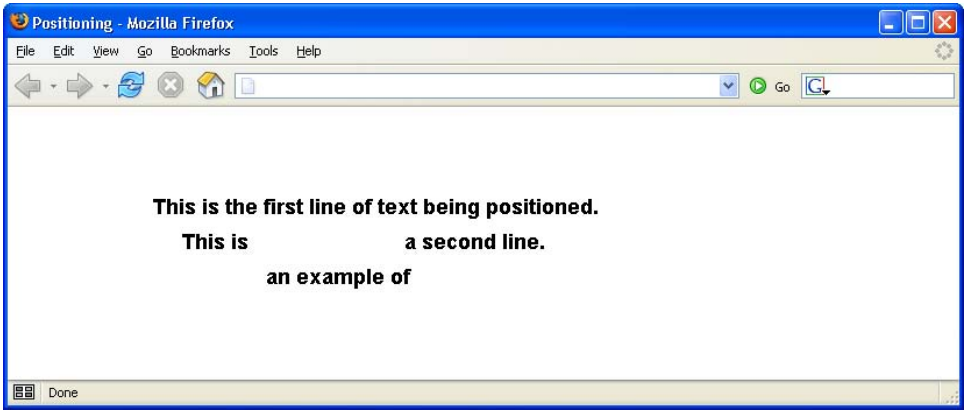
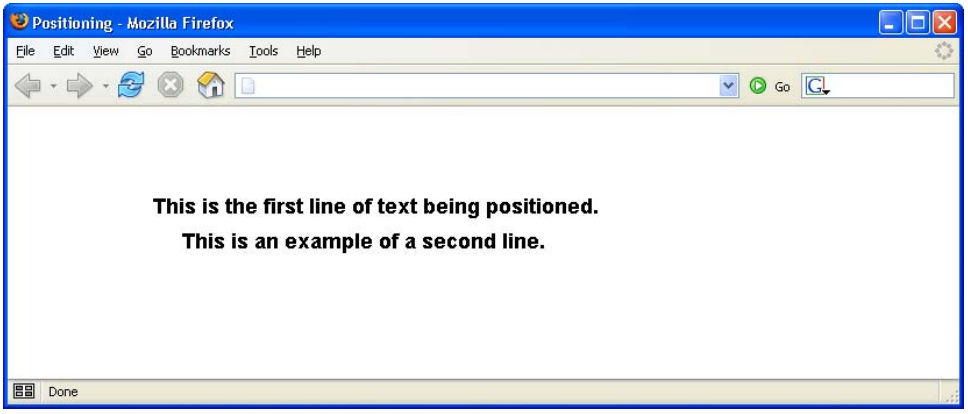


Figure 8.7. The same example with the positioning removed



Don't worry if this concept still seems a bit confusing; we'll be looking at how these concepts work in practice as we create our layouts.

The Box Model

From the perspective of a style sheet, every item you deal with in an HTML page can be viewed as existing inside a box. This fact is generally far more obvious when you're formatting large chunks of content, like the three main page areas we've identified in our design. But it's true even when you're dealing with individual components of those elements, like headings, lists, list elements, and even segments of text.

The basic CSS box model is shown in Figure 8.8.

Figure 8.8. The basic CSS box model

At the center of the CSS box model is the content itself. Don't think of this "content" as being the same as words or images that might comprise the content of a news story or a set of links. "Content" describes any item that's contained within the area of the box.

Notice from the diagram that the visible width of the box is determined by adding together the content width, the padding, and the border. The margin determines the distance between each side of the visible box and adjacent elements. Similarly, the visible height of the box is determined by adding the height of the content to the padding and border settings. Once again, the margin determines how far the box will be separated from adjacent objects vertically.

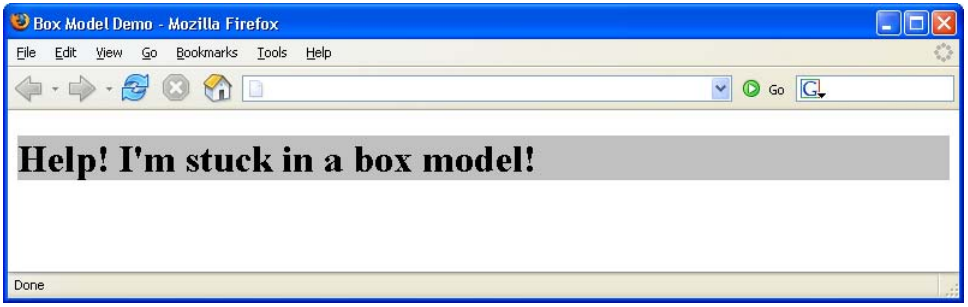
The width of each of these elements—margin, border, and padding—can be set using four CSS properties (one for each side of the box), or a single shorthand property. Border behavior is slightly more complicated because, in addition to width, a border can have characteristics such as line style and color.

In this discussion, I'll begin by explaining and demonstrating the use of padding in some detail. Then, I'll move on to a discussion of margins, which will be briefer, as it's so similar to padding. Finally, I'll discuss borders.

Order the print version of this book to get all 500+ pages!

For the next few sections, I'll use a basic, single-box layout to demonstrate CSS rule techniques. It starts out as the layout shown in Figure 8.9, with no padding, border, or margin: the content is the same size as the box.

Figure 8.9. Starting point for the box model demonstration



I've given the `h1` element a gray background so you can see more easily the impact of the effects I'll be demonstrating. The HTML below produces the page shown in Figure 8.9:

File: **boxmodel.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Box Model Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
      h1 {
        background-color: #c0c0c0;
        color: black;
      }
    </style>
  </head>
  <body>
    <h1>Help! I'm stuck in a box model!</h1>
  </body>
</html>
```

Throughout the rest of this discussion, I'll be modifying only the style sheet information, so I'll reproduce only that section of the code, indicating any changes in bold.

Pixels vs Percentages

As the box model deals with the display of content on the screen, the pixel is the most commonly used of the absolute measurement units in CSS. However, if you need to create a layout that takes up all of the available space, regardless of how big the browser window is, it's necessary to use the percentages rather than pixels. Such layouts are characterized by their "stretchy" behavior—the page elements expand and contract proportionately as the user resizes the browser window.

Padding Properties

Four properties together define the padding around an object in a CSS rule: `padding-left`, `padding-right`, `padding-top`, and `padding-bottom`.

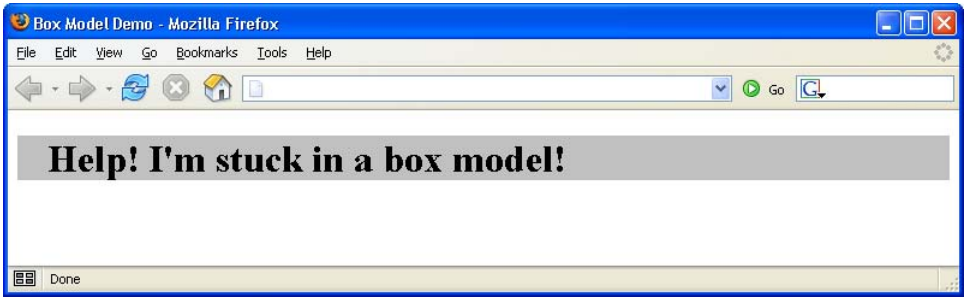
Let's change just one of the padding settings to get a feel for how this works. Modify the style sheet in the sample file, so that it replicates the following fragment (remember that the new material is presented in bold text below):

File: **`boxmodel.html` (excerpt)**

```
h1 {
  background-color: #c0c0c0;
  color: black;
  padding-left: 25px;
}
```

The result of this change is shown in Figure 8.10. Notice that the text now begins 25 pixels from the left side of the box, resulting in 25 pixels of blank, gray space to the left of the text.

Figure 8.10. Demonstrating `padding-left`



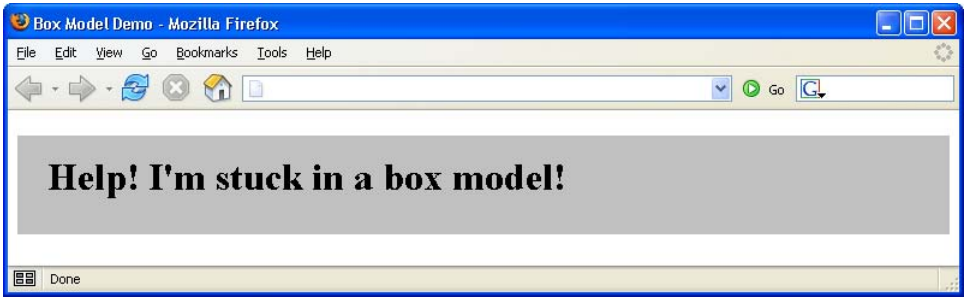
Order the print version of this book to get all 500+ pages!

As you'd expect, you can set the other padding sizes the same way, as this code fragment shows:

File: **boxmodel.html (excerpt)**

```
h1 {
  background-color: #c0c0c0;
  color: black;
  padding-left: 25px;
  padding-top: 15px;
  padding-bottom: 30px;
  padding-right: 20px;
}
```

Figure 8.11. Defining all four padding properties



You can see the effects of these changes in Figure 8.11.

You may notice that the padding on the right-hand side appears not to have worked. You asked for 20 pixels, but no matter how wide you stretch the window, the gray area that defines the box containing our `h1` element just goes on and on.

This is because `padding-right` creates a space between the right edge of the text and the right edge of the heading, as represented by the gray box. The spacing is difficult to see in this case, because the heading automatically spans the width of the browser window, leaving plenty of room for the text to breathe on the right-hand side. If you make the browser narrow enough, though, you can see the padding take effect.

Figure 8.12. Demonstrating the effect of `padding-right`

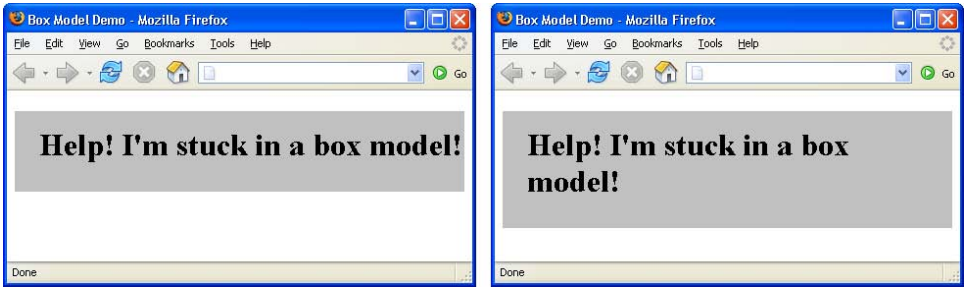


Figure 8.12 demonstrates this principle. The first screenshot shows how the page from Figure 8.11 looks if you narrow the browser window so that there would be room for the word "in" on the first line if `padding-right` was not set as it is. The second screenshot reinforces this idea by showing the page resized so that one word only fits on each line. Notice that, in several cases, the right padding size looks large enough to accommodate the word on the next line. In fact, merely removing the `padding-right` declaration from the style sheet produces the result shown in Figure 8.12.

Because it's often necessary to adjust padding around objects in HTML, the CSS standards define a shorthand property that's simply called `padding`. You can give this property up to four values; Table 8.1 identifies how the properties will be assigned in each case.

Order the print version of this book to get all 500+ pages!

Table 8.1. Effects of multiple values on `padding` shorthand property

Number of Values	Interpretation
1	Set all four padding values to this value.
2	Set the top and bottom padding to the first value, and left and right padding to the second.
3	Set the top padding to the first value, right and left to the second value, and bottom to the third value.
4	Set the top padding to the first value, right padding to the second, bottom padding to the third, and left padding to the fourth value.

Remembering the Order

To remember the order in which these values are specified, simply recall that they're identified in clockwise order from the top, or remember the mnemonic *trouble* (top, right, bottom, and left).

For example, the style rule above could be rewritten using the padding shorthand property as follows:

File: **`boxmodel.html` (excerpt)**

```
h1 {
  background-color: #c0c0c0;
  color: black;
  padding: 15px 20px 30px 25px;
}
```

To create equal top and bottom padding, and equal left and right padding, you could use:

File: **`boxmodel.html` (excerpt)**

```
h1 {
  background-color: #c0c0c0;
  color: black;
  padding: 15px 25px;
}
```

Finally, to create equal padding on all four sides of the `h1` element, you could use this markup:

File: **boxmodel.html (excerpt)**

```
h1 {
  background-color: #c0c0c0;
  color: black;
   padding: 25px;
}
```

What would happen if you used either ems or percentages for the padding values? The two units have slightly different effects: the em unit scales the padding according to the size of the font of the content, while the percentage unit scales the padding according to the width or height of the block that contains the element. To demonstrate these effects, let's work with a new HTML page that displays two headings against colored backgrounds on a page of a contrasting color.

Here's the HTML for that demonstration page:

File: **boxmodel2.html**

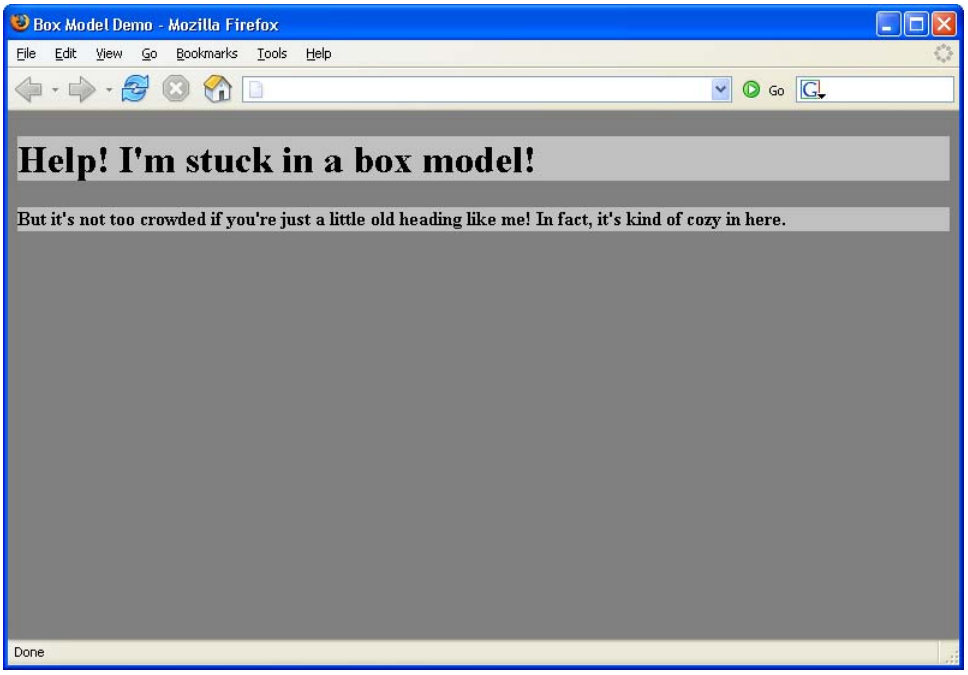
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Box Model Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
      body {
        background-color: #808080;
        color: black;
      }
      h1, h4 {
        background-color: #c0c0c0;
        color: black;
      }
    </style>
  </head>
  <body>
    <h1>Help! I'm stuck in a box model!</h1>
    <h4>But it's not too crowded if you're just a little old
        heading like me! In fact, it's kind of cozy in here.</h4>
  </body>
</html>
```


Order the print version of this book to get all 500+ pages!

Notice that I've given the page a dark grey background, and I've added an `h4` element, which I've styled in the same CSS rule as the `h1` element.

This HTML page displays as shown in Figure 8.13.

Figure 8.13. Proportional `padding` page starting point



Now, let's change the style sheet for this page so that it uses the `padding` property to create a single-em padding space around the objects. The following code fragment will do the trick:

File: **`boxmodel2.html` (excerpt)**

```
body {
  background-color: #808080;
  color: black;
}
h1, h4 {
  background-color: #c0c0c0;
  color: black;
  padding: 1em;
}
```

As you can see in Figure 8.14, the amount of padding that appears around the two heading elements is proportional to the size of the font used in the elements themselves.

em: a Height Measurement

Remember that one em is equal to the height of the font in use. Consequently, much more space is placed around the `h1` element than around the `h4` element.

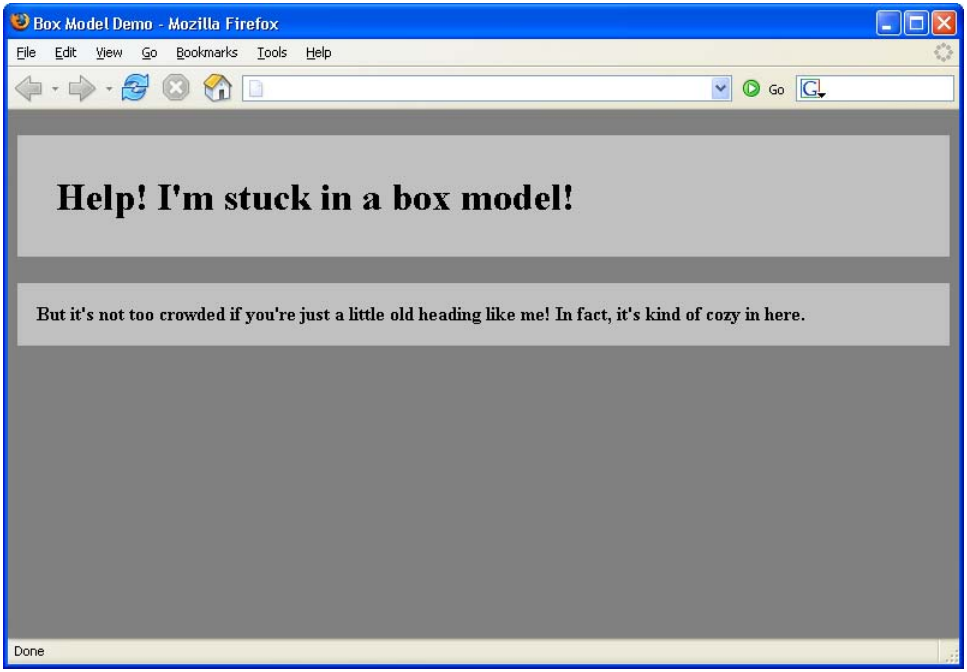
Let's see what happens if we use a percentage, rather than an em, for the proportional padding value. Change the HTML so that the style sheet looks like this:

File: **boxmodel2.html (excerpt)**

```
body {
  background-color: #808080;
  color: black;
}
h1, h4 {
  background-color: #c0c0c0;
  color: black;
  padding: 10%;
}
```

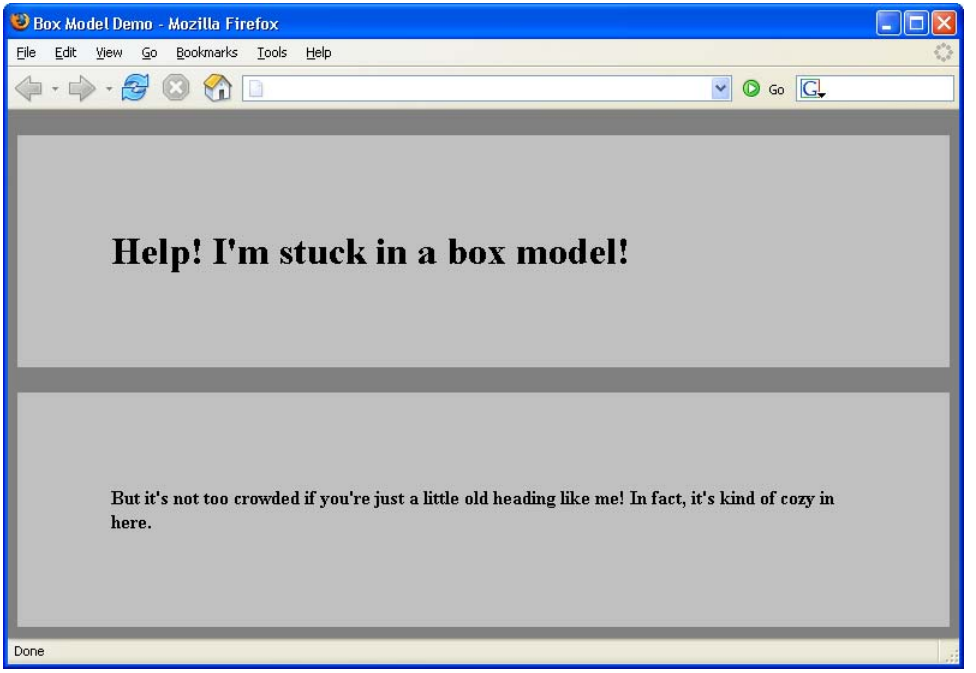

Order the print version of this book to get all 500+ pages!

Figure 8.14. Using ems for proportional padding



The result of this change can be seen in Figure 8.15. Wow! There's a huge amount of space around those elements. The browser has applied 10% of the width of the page as padding on all four sides.

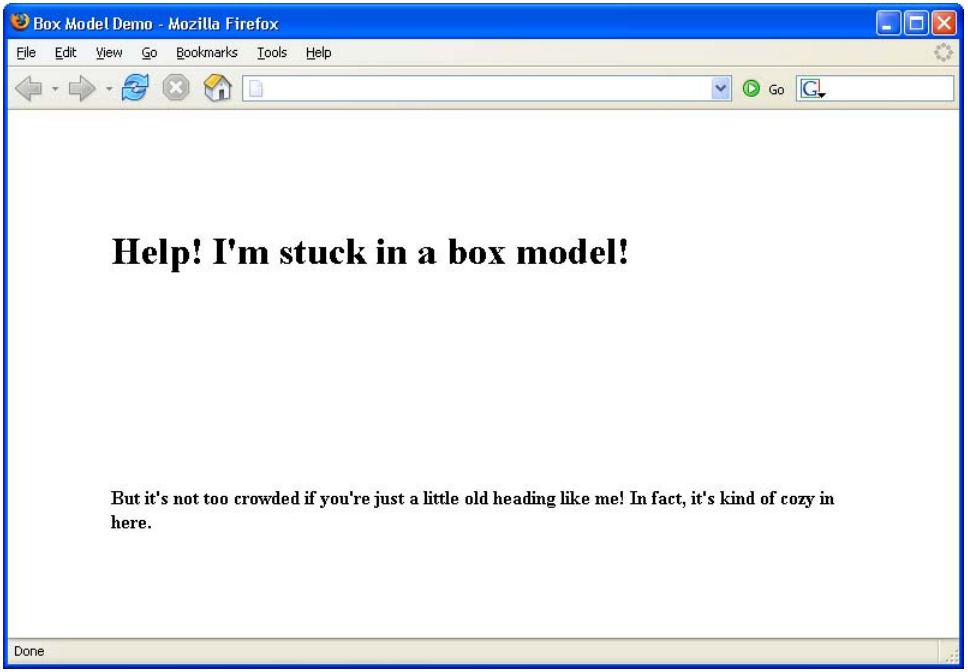
Figure 8.15. Using percentage for proportional spacing



I've been using a background color behind the text of these elements to make it easy to see the effect of the different padding settings, but the background colors aren't required. Figure 8.16 uses the same HTML code as Figure 8.15; the only difference is that I've removed the background colors from the `body`, `h1`, and `h4` elements. As you can see, these elements maintain their relative spacing.

Order the print version of this book to get all 500+ pages!

Figure 8.16. Demonstrating `padding` without colored backgrounds



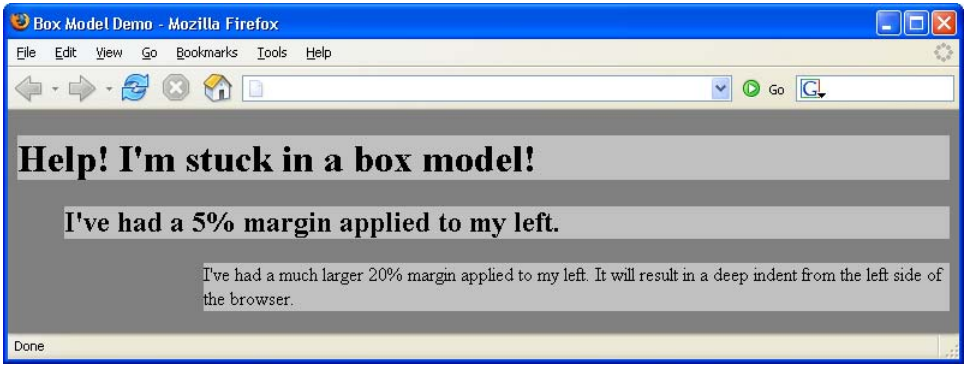
Margin Properties

The difference between margins and padding is that margins exist outside the boundaries of the object, while padding exists inside those boundaries. Figure 8.17 illustrates this difference according to the style sheet rules that are set in the code fragment below. Margins are set in the same way as padding; the only difference is the substitution of the word "margin" for the word "padding."

```
body {
  background-color: #808080;
  color: black;
}
h1 {
  background-color: #c0c0c0;
  color: black;
}
h2 {
  background-color: #c0c0c0;
  color: black;
```

```
  margin-left: 5%;
}
p {
  background-color: #c0c0c0;
  color: black;
  margin-left: 20%;
}
```

Figure 8.17. `margin-left` settings pushing the content and background right



Notice that the second-level heading and the paragraph, both of which have `margin-left` properties, are indented from the left edge of the browser. But, unlike the example in which we set the `padding-left` property, the text and its background color block are indented in this case. This is because the padding, the color block, and the text are inside the content box, while the margin is outside that box.

Next, let's apply `padding-left` and `margin-left` settings to the code fragment:

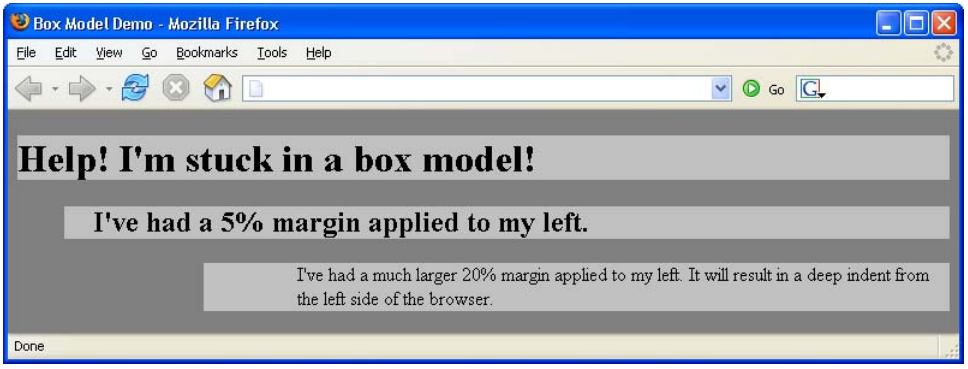
```
body {
  background-color: #808080;
  color: black;
}
h1 {
  background-color: #c0c0c0;
  color: black;
}
h2 {
  background-color: #c0c0c0;
  color: black;
```


Order the print version of this book to get all 500+ pages!

```
  margin-left: 5%;
  padding-left: 1em;
}
p {
  background: #c0c0c0;
  color: black;
  margin-left: 20%;
  padding-left: 10%;
}
```

As you can see in Figure 8.18, the above markup has caused the margin to push the HTML elements and their surrounding background color blocks to the right, while the padding has moved the text to the right within the colored background blocks.

Figure 8.18. Combining `margin-left` with `padding-left`



If you load the above HTML (from the file included in the code archive for this book) and resize it, you'll notice that the indentation of the paragraph and the heading changes as the width of the window changes. That's because we used relative values of 20% for the margin and 10% for the padding. Both of these values are calculated relative to the width of the containing block, which in this case is the browser window. The bigger the browser window, the bigger the margin and padding on the paragraph. The padding on the heading doesn't change, as it's specified in ems.

Margins, Padding, and Lists

By default, all visual browsers will apply a 50-pixel margin to the left edge of a list. This allows room for the list item markers (bullets in the case of a bulleted

list; numbers in the case of an ordered list). Unfortunately, the CSS Specification doesn't say explicitly whether this space should be implemented as left margin or left padding in the browser's default style rules. However, the description of the `marker-offset` property does imply that margin is the way to go.

Whatever the intent of the specification, Firefox and Safari apply a default padding to the left side of lists, while most other browsers (including Internet Explorer and Opera) use a margin. You can test this easily by applying a `background-color` to an `ol` or `ul` element. On most browsers, the background will not cover the list item markers; on Firefox and Safari, they will.

For this reason, whenever you apply your own left margin or padding value to a list, you must be sure to specify both. If you applied only a margin, for example, the default list indentation would display in Firefox, but be overridden on all other browsers. If you applied a padding value only, the default 50-pixel margin would display on Internet Explorer. Only by specifying both margin and padding (usually by setting `padding: 0` and using `margin` to do the job) can you ensure consistent rendering across current browsers.

You can set vertical margins with the `margin-top` and `margin-bottom` properties. Here's another HTML page that demonstrates vertical margins:

File: **boxmodel3.html**

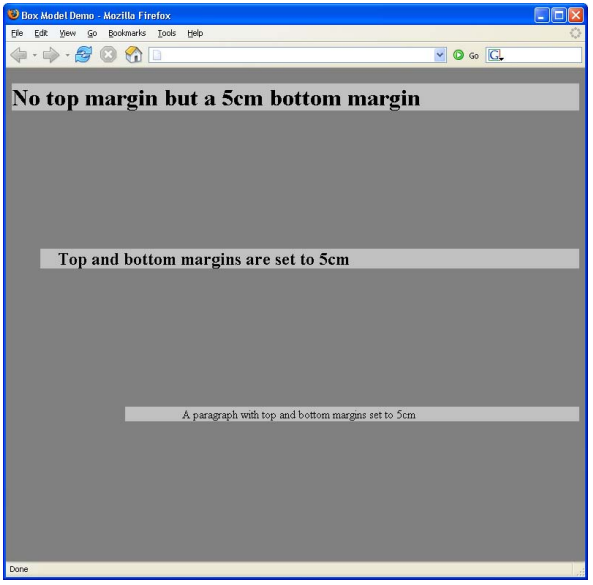
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Box Model Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
      body {
        background-color: #808080;
        color: black;
      }
      h1 {
        background-color: #c0c0c0;
        color: black;
        margin-bottom: 5cm;
      }
      h2 {
        background-color: #c0c0c0;
        color: black;
        margin-left: 5%;
```


Order the print version of this book to get all 500+ pages!

```
        margin-top: 5cm;
        margin-bottom: 5cm;
        padding-left: 1em;
      }
      p {
        background: #c0c0c0;
        color: black;
        margin-left: 20%;
        padding-left: 10%;
        margin-top: 5cm;
        margin-bottom: 5cm;
      }
    </style>
  </head>
  <body>
    <h1>No top margin but a 5cm bottom margin</h1>
    <h2>Top and bottom margins are set to 5cm</h2>
    <p>A paragraph with top and bottom margins set to 5cm</p>
  </body>
</html>
```

This page renders as shown in Figure 8.19.

Figure 8.19. Demonstrating vertical margins



Unlike horizontal margins, vertical margins are not cumulative. If you have two elements stacked one atop the other, like the `h1` and `h2` elements shown in Figure 8.19, the vertical spacing between them will be the greater of the `margin-bottom` setting of the top element, and the `margin-top` setting of the bottom element. In this case, they are both `5cm`, so the distance between the two elements is `5cm` (not `10cm`, as you might have supposed). If I had defined the `margin-bottom` of the `h1` as `10cm`, then the vertical distance separating the two elements would have been `10cm`. The containing block in this case is the `body`, which is, for all practical purposes, the same as the browser window's client area.

It is possible to use negative values for margin property settings. This comes in handy when you've set a `margin-left` property for the `body` of an HTML page, but you want to move an element closer to the left margin of the page. The following HTML results in the display shown in Figure 8.20:

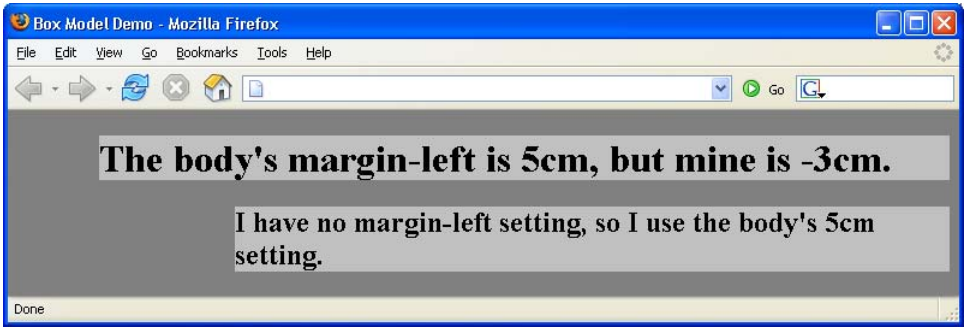
File: **boxmodel4.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Box Model Demo</title>
    <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1" />
    <style type="text/css">
      body {
        background-color: #808080;
        color: black;
        margin-left: 5cm;
      }
      h1 {
        background-color: #c0c0c0;
        color: black;
        margin-left: -3cm;
      }
      h2 {
        background-color: #c0c0c0;
        color: black;
      }
    </style>
  </head>
  <body>
    <h1>The body's margin-left is 5cm, but mine is -3cm. </h1>
    <h2>I have no margin-left setting, so I use the body's 5cm
        setting.</h2>
```


Order the print version of this book to get all 500+ pages!

```
  </body>
</html>
```

Figure 8.20. Negative margin setting in practice



As with the `padding` property, the `margin` shorthand property lets you set all four margins with a single declaration, and interprets multiple values using the rules shown in Table 8.1.

Border Properties

Border properties are more complex than padding and margin properties because they affect not only the spacing between objects, but the appearance of that intervening space. A border can be, and usually is, visible. In most ways, managing border properties is similar to the process for managing margins and padding, but there are some key differences.

Borders have three types of properties: style, width, and color. By default, a border's style is set to `none`, its width to `medium`,[1] and its color to the text color of the HTML element to which it is applied.

The `border-style` property can take any one of a range of constant values. The available values are `solid`, `dashed`, `dotted`, `double`, `groove`, `ridge`, `inset`, `outset`, `hidden`, and `none`.

The `hidden` value has the same effect as `none`, except when applied to `table` layouts. Refer to the `border-style` property in Appendix C for further details.

[1]Netscape 4 sets a default border width of `0`, so you can't rely on the default value if you wish to target that browser.

W3C specifications largely leave the issue of the precise appearance of these borders up to the browsers, so don't be surprised if the results of using these characteristics vary a bit from browser to browser, and platform to platform. But, as is the case with default behaviors for other border settings, generally speaking, the browsers treat this issue predictably and satisfactorily within reason.

The width of a border around an object can be set either with four individual declarations, or with the `border-width` shorthand syntax. The four properties are `border-top-width`, `border-right-width`, `border-bottom-width`, and `border-left-width`. Each of these properties can be set with an absolute or relative length unit (such as pixels, ems, percentages, or inches), or with one of three descriptive settings: `thin`, `medium`, or `thick`.

If you use the descriptive settings of `thin`, `medium`, and `thick`, the results are browser-dependent. However, they are fairly predictable and consistent across browsers and operating systems, within a pixel or so for each of the three descriptive settings.

Specific Border Measurements

If you wish to use specific measurements for border widths, you should use pixels. This is the most meaningful unit of measurement for screen layouts, which is where `border-width` is an important property.

You can control the colors associated with all four borders using the `border-top-color`, `border-right-color`, `border-bottom-color`, and `border-left-color` properties, or you can just use the `border-color` shorthand property.

As we discovered in Chapter 5, you can supply a color argument in any of the standard ways: using a hexadecimal RGB code (as in `#ff9900`), using a three-digit hexadecimal RGB shortcut (as in `#f90`), via the `rgb` function (as in `rgb(102,153,0)`), or using a standard color name (as in `red`).

The shorthand properties `border-style`, `border-width`, and `border-color` all accept multiple values.

There is one additional shorthand property that's probably the most widely used. The `border` property allows you to specify the style, width, and color of all four borders of an object in a compact form. Since a border that's uniform on all sides is most often your desire, this is an efficient way to set border property values.

Order the print version of this book to get all 500+ pages!

The following style rule will produce a uniform, three-pixel, solid, red border around any element with a `class="warning"`:

```
.warning {
  border: 3px solid red;
}
```

Constructing the Layout

Now that we have some background knowledge of the ways in which elements behave when they're positioned using CSS, we can put our learning into practice with our first layout.

Create a new style sheet named `styles.css` and link it to the Footbag Freaks document we created earlier by adding the following markup to the `head` of the document:

File: **index.html (excerpt)**

```
<head>
  <title>Footbag Freaks</title>
  <meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
  <link rel="stylesheet" type="text/css" href="styles.css" />
</head>
```

The first element to which we'll add CSS is the `body` element. The design has a background image that starts with a pattern but gradually blends into a deep blue. To create this effect on our page, we'll apply the image as a tiled background, and give the page a blue background color. This way, when the background image finishes, it seamlessly merges into the blue page background.

Download Footbag Freaks

The Footbag Freaks web site, including all images, is available for download as part of the code archive for this book.

Let's also set a font family and size, and set the margin and padding for the page (the space between the edge of the viewport and your content) to `0`, using the markup below.

File: **styles.css**

```
body {
  margin: 0;
```

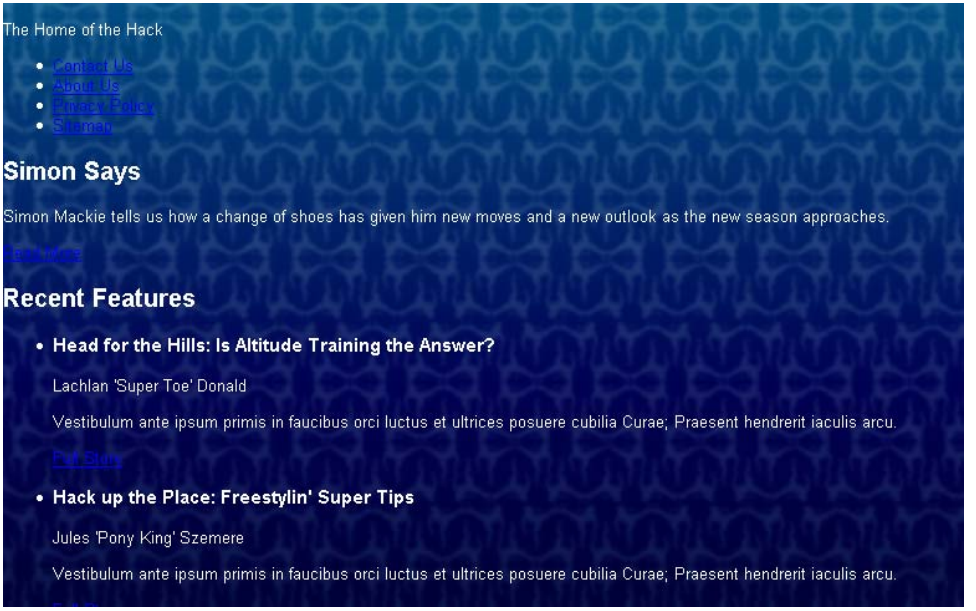
```
  padding: 0;
  background-color: #050845;
  color: white;
  background-image: url(img/bg.jpg);
  background-repeat: repeat-x;
  font: small Arial, Helvetica, Verdana, sans-serif;
}
```

Setting Freaks `font-size`

I've set the `font-size` on the `body` using the keyword `small`. As we create the rest of the style sheet, I'll use percentage font sizes to make the size of each element a percentage of `small`.

Now, your background image should tile across the width of the page, as shown in Figure 8.21.

Figure 8.21. The background image tiling across the width of the page



In our layout image, the content of the page is contained in an off-white box. To create this box, we need to add another `div` in which we can wrap the content.

Order the print version of this book to get all 500+ pages!

So, immediately after the opening `<body>` tag in your document, add the markup shown in bold below:

File: **index.html (excerpt)**

```
<body>
  <div id="wrapper">
    <div id="header">
      <p>The Home of the Hack</p>
```

Don't forget to close this `div` immediately before the document's closing `</body>` tag, like so:

File: **index.html (excerpt)**

```
      <p><a href="">more</a></p>
    </div> <!-- main -->
  </div> <!-- wrapper -->
</body>
```

Now, let's add to the style sheet the rules that will give the box an off-white background. We'll also insert rules that add a margin to the wrapper area, creating a space between the wrapper and the `body` element to let the background image show through:

File: **styles.css (excerpt)**

```
#wrapper {
  background-color: #fdf8f2;
  color: black;
  margin: 30px 40px 30px 40px;
}
```

Figure 8.22. The effect of the styled wrapper `div`

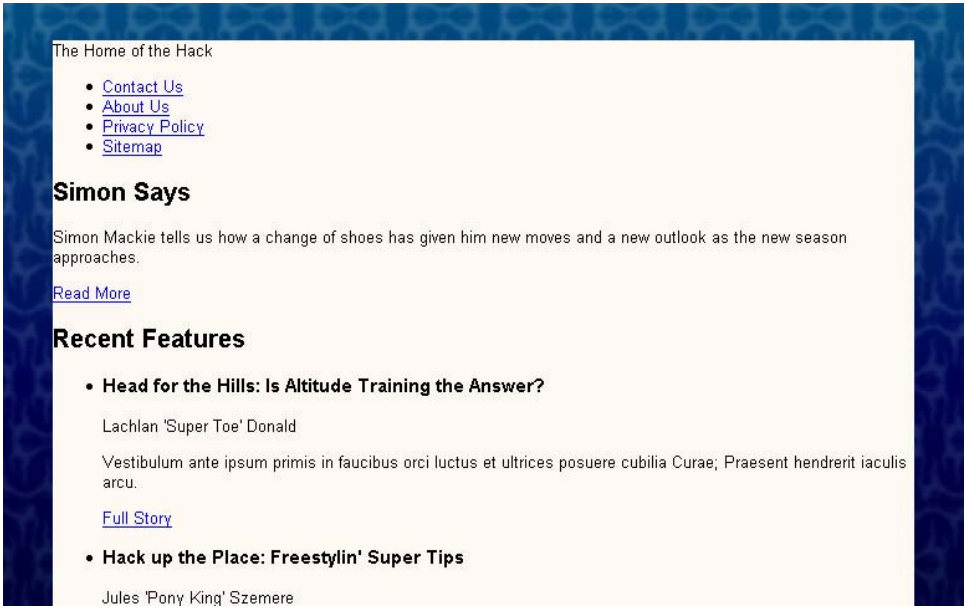


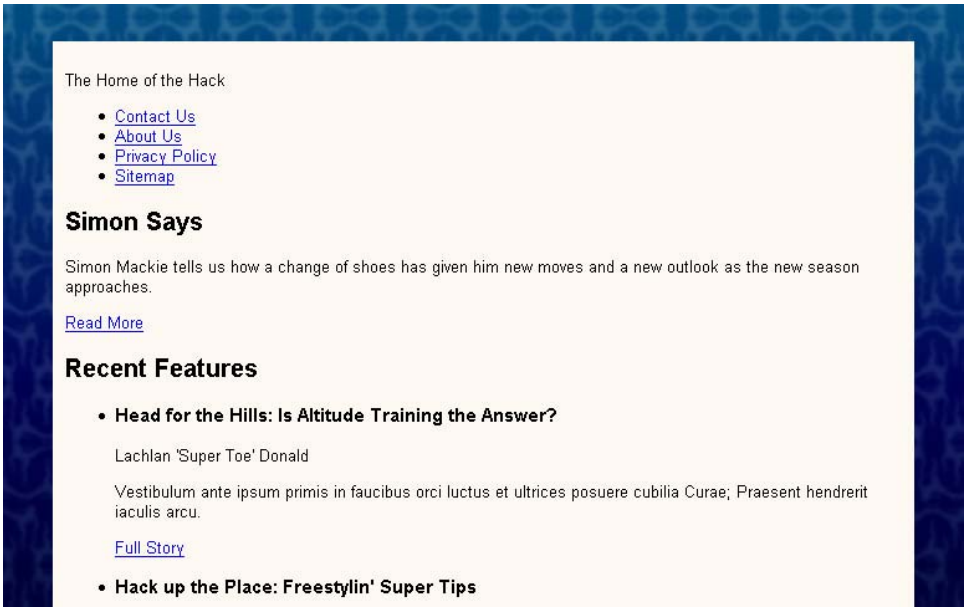
Figure 8.22 shows the results of our work. The margin has created a space that lets the background show through, but the content inside the wrapper bumps right up against the edge of the off-white area. We can create some extra space here by adding padding to the `#wrapper` rule, as shown in the markup below. The resulting display is shown in Figure 8.23.

File: **`styles.css` (excerpt)**

```
#wrapper {
  background-color: #fdf8f2;
  color: black;
  margin: 30px 40px 30px 40px;
  padding: 10px;
}
```


Order the print version of this book to get all 500+ pages!

Figure 8.23. Extra padding creating space between the box's edge and its content



The Header Area

Let's turn our attention to the header area of our layout, which contains the site logo and main navigation. You'll remember that when we created our HTML document, we didn't add any images: we were going to decide how best to include our images as we developed the layout. But now, let's add the logo image using the `img` element. We'll also include the site name as `alt` text for the image, so that users who are browsing the site with images turned off, and those with screen readers, can read the name of the site.

In your document, insert the image directly below the opening `header div`, like this:

File: **`index.html` (excerpt)**

```
<body>
  <div id="wrapper">
    <div id="header">
      <img src="img/logo.gif" alt="Footbag Freaks" height="77"
          width="203" />
```

```
      <p>The home of the hack</p>
      <ul>
        <li><a href="">Contact Us</a></li>
        <li><a href="">About Us</a></li>
        <li><a href="">Privacy Policy</a></li>
        <li><a href="">Sitemap</a></li>
      </ul>
    </div> <!-- header -->
```

If you view the page in a browser, you should see the image in the top, left corner of the off-white box.

The graphic for our page layout shows a thin, light-blue border that appears above and below the site's tagline and navigation. How will we create this effect? Let's contain the tagline and navigation in another `div` to which we can apply a top and bottom border. Add the `div` like so:

File: **index.html (excerpt)**

```
<body>
  <div id="wrapper">
    <div id="header">
      <img src="img/logo.gif" alt="Footbag Freaks" height="77"
          width="203" />
      <div id="header-bottom">
        <p>The home of the hack</p>
        <ul>
          <li><a href="">Contact Us</a></li>
          <li><a href="">About Us</a></li>
          <li><a href="">Privacy Policy</a></li>
          <li><a href="">Sitemap</a></li>
        </ul>
      </div> <!-- header-bottom -->
    </div> <!-- header -->
```

We can now address `#header-bottom` as we add the top and bottom borders:

File: **styles.css (excerpt)**

```
#header-bottom {
  border-top: 1px solid #b9d2e3;
  border-bottom: 1px solid #b9d2e3;
}
```

To style the navigation list and tagline, we'll use some simple text formatting properties that should now be fairly familiar!

Order the print version of this book to get all 500+ pages!

Example 8.30. `styles.css` (excerpt)

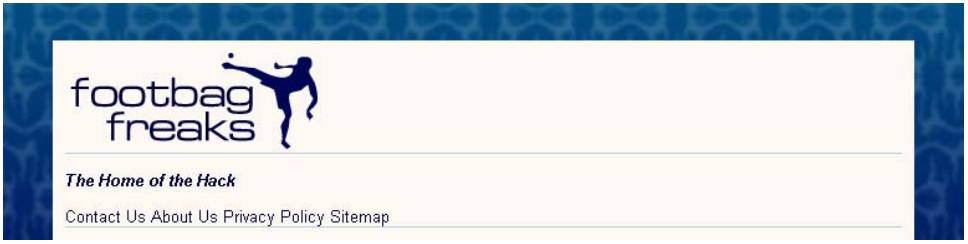
```
#header-bottom ul {
  margin: 0;
  padding: 0;
}
#header-bottom li {
  display: inline;
}
#header-bottom a:link, #header-bottom a:visited {
  text-decoration: none;
  background-color: #fdf8f2;
  color: #050845;
}
#tagline {
  font-weight: bold;
  background-color: #fdf8f2;
  color: #050845;
  font-style: italic;
}
```

We also need to add an `id` attribute to the paragraph that contains our tagline:

File: **`index.html` (excerpt)**

```
<p id="tagline">The home of the hack</p>
```

Figure 8.24. Styling navigation list items with `display: inline`



We set the `margin` and `padding` on the list within this area to `0`, then set the `li` element's `display` property to `inline`, which will cause the list items to display on the same line, rather than having each item display on a new line. Figure 8.24 shows this effect in action. We also styled the navigation links—again using the dark blue and removing the underlines from them—and the tagline, which we made bold, italic, and the same blue as our navigation items.

The problem with the display shown in Figure 8.24 is that it's difficult to distinguish the links in the navigation list from one another. The recommended solution[2] to this problem is to add a visible character—such as the pipe character (|)—between each of the links, as I've done in the markup below:

File: **index.html (excerpt)**

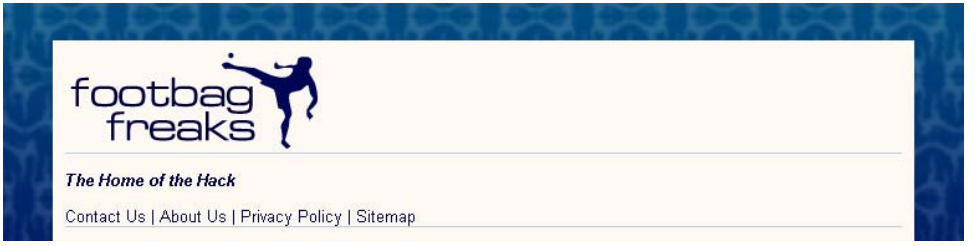
```
<ul>
  <li><a href="">Contact Us</a> | </li>
  <li><a href="">About Us</a> | </li>
  <li><a href="">Privacy Policy</a> | </li>
  <li><a href="">Sitemap</a></li>
</ul>
```

We can also set the color of the list items to dark blue (`#050845`), so that the pipe character that sits outside of the anchor element will be blue, too. Our refined header design is shown in Figure 8.25.

Example 8.33. `styles.css` (excerpt)

```
#header-bottom li {
  display: inline;
  background-color: #fdf8f2;
  color: #050845;
}
```

Figure 8.25. After styling the text elements in the header area



The header is really starting to take shape now! Our next step is to move the tagline and navigation up onto the same line. To do this, we'll have to use a property that, while we haven't discussed it in detail yet, will become more important to us as we progress through these layouts. That property is `float`.

[2]This recommendation was made as part of the Web Content Accessibility Guidelines (WCAG) 1.0. The checkpoint that covers this specific issue can be seen at http://www.w3.org/TR/WCAG10/wai-pageauth.html#tech-divide-links.

Order the print version of this book to get all 500+ pages!

The `float` Property

`float` is one of the most interesting and often-used CSS properties. It takes a value of `left`, `right`, or `none` (though `none`, the default, is rarely used). `float` forces the element to which it's applied to display outside its natural position in the containing box; a `float` value of `left` or `right` pushes the element to the left or the right of its natural position, respectively. This property can be used within any block element.

The `float` property is designed to replace the `align` attribute that's associated with the HTML `img` element, and has, for all practical purposes, precisely the same effect. The `align` attribute is deprecated in favor of the `float` property in recent releases of HTML Recommendations from the W3C. The following HTML fragment uses the `float` property to produce the result shown in Figure 8.26:

```
<p><img src="logo.gif" alt="Footbag Freaks Logo"
    width="203" height="77"
    style="float: left; padding-right: 1em;" />The Footbag Freaks
    logo appears to the left of this paragraph. Depending on
    whether or not I use the CSS <code>float</code> property, I
    may see more than one line of text beside the logo. The CSS
    <code>float</code> property replaces the deprecated
    <code>align</code> attribute of the HTML <code>img</code>
    element and has an identical effect.</p>
```

Figure 8.26. Achieving image-text alignment using the CSS `float` property

The Footbag Freaks logo appears to the left of this paragraph. Depending on whether or not I use the CSS `float` property, I may see more than one line of text beside the logo. The CSS `float` property replaces the deprecated `align` attribute of the HTML `img` element and has an identical effect.

The `float` property has one major advantage over the `align` attribute: `float` can be applied to elements other than images, whereas application of the old `align` attribute was limited to `img`, `applet`, and `object` elements.

No Dimensions? Declare a `width`

When using the `float` property on elements that don't have well-defined dimensions, you must include a `width` declaration in your CSS. An `img` is an example of an element with well-defined dimensions, whereas a paragraph, a heading, or a `div` doesn't.

Using `float` in our Header

We'll be exploring the `float` property in more detail in the next chapter, when we create a layout that relies on `float` for the positioning of the page's main sections. However, at this point we can use our knowledge of `float` to align the tagline and navigation correctly. The element that we're going to float is the tagline paragraph, so add the rules marked in bold below to your tagline rule:

Example 8.34. `styles.css` (excerpt)

```
#tagline {
  font-weight: bold;
  background-color: #fdf8f2;
  color: #050845;
  font-style: italic;
  margin: 0;
  padding: 0 0 0 20px;
  width: 300px;
  float: left;
}
```

We set `float` to `0` so that the paragraph's default margin is removed. We then add 20 pixels of left padding to move the tagline in from the left-hand side, and give it a width of 300 pixels to provide a bit of space to its right, as is indicated in the page's original layout graphic. We then set the value of `float` to `left`, so it sits to the left of the rest of the content, which in this case, is our navigation list.

After making this change to the rules for the tagline paragraph, save your style sheet and view your page in a browser. You should see the navigation display alongside the tagline. These elements behave in exactly the same way as the paragraph that wraps around the image in the example we discussed above. All we need to do now is to align the list of navigation items to the right, and alter the padding on the list to move it in slightly from the right-hand edge. Here's the markup you'll need; the resulting display is depicted in Figure 8.27.

File: **`styles.css` (excerpt)**

```
#header-bottom ul {
  margin: 0;
  padding: 0 30px 0 0;
  text-align: right;
}
```


Order the print version of this book to get all 500+ pages!

Figure 8.27. The display after floating the tagline and aligning the navigation

The final task that will complete the heading is to add the little footbag image that displays to the right of the navigation in our layout image. First, add the actual image to your document, beneath the navigation list. In the markup below, I gave this image an empty `alt` attribute, so that nothing about this image would be read out by a screen reader—this image is included for display purposes only. I've also given the image an `id` of `ball`.

File: **`index.html` (excerpt)**

```
<div id="header">
  <img src="img/logo.gif" alt="Footbag Freaks" height="77"
      width="203" />
  <div id="header-bottom">
    <p id="tagline">The home of the hack</p>
    <ul>
      <li><a href="">Contact Us</a> | </li>
      <li><a href="">About Us</a> | </li>
      <li><a href="">Privacy Policy</a> | </li>
      <li><a href="">Sitemap</a></li>
    </ul>
    <img src="img/header-ball.gif" height="24" width="20" alt=""
        id="ball" />
  </div> <!-- header-bottom -->
</div> <!-- header -->
```

Now, let's use our first bit of absolute positioning in the CSS to get the image to line up properly. We know the location at which the image should be positioned relative to the top and right-hand sides of the document, as we know the height of the logo and width of the margin on the wrapper `div`. The following CSS will place the ball in the correct position at the end of the navigation:

```
#ball {
  position: absolute;
  top: 110px;
```

```
  right: 55px;
}
```

The header section is now complete! It's displayed in Figure 8.28.

Figure 8.28. The completed header section of the layout

The Content Area

Let's move on to create the look and feel of the main content area of the page. The first thing we'll do is contain the `sidebar` and `content` `div`s within another `div` that has an `id` of `main`. This will help us to line up the `sidebar` and `content` `div`s beneath the header. Add the opening `<div id="main">` just after the header's closing `</div>`:

File: **index.html (excerpt)**

```
    <img src="img/header-ball.gif" height="24" width="20" alt=""
        id="ball" />
  </div> <!-- header-bottom -->
</div> <!-- header -->
<div id="main">
  <div id="content">
    <h2>Simon Says</h2>
```

Close this `div` immediately after the closing `</div>` tag of the `sidebar` `div`. In the style sheet, give `#main` a `margin-top` of ten pixels to separate the content and header areas, as shown in the snippet below. We'll return to `#main` later, as we create our layout.

Order the print version of this book to get all 500+ pages!

File: **styles.css (excerpt)**

```
#main {
  margin-top: 10px;
}
```

Now, let's create a rule for `#content`. Add the following set of declarations to your style sheet:

File: **styles.css (excerpt)**

```
#content {
  margin: 0 240px 0 0;
  border: 1px solid #b9d2e3;
  background-color: white;
  color: black;
}
```

We set the top margin of `#content` to `0`. Then, we add a 240-pixel right-hand margin, leaving space for us to position our sidebar later.

We also give the box a solid, single-pixel border in the same blue we used for the heading borders, and give it a background color of white.

The Main Feature

At the very top of the page is a "boxout": an area that's visually contained within a box that highlights it. This particular boxout highlights the main feature article. Let's look at that now.

Create a container for the main feature area by adding a `div` with an `id` of `mainfeature`; wrap it around the heading, paragraph, and link of the main feature:

File: **index.html (excerpt)**

```
<div id="content">
  <div id="mainfeature">
    <h2>Simon Says</h2>
    <p>Simon Mackie tells us how a change of shoes has given him
        new moves and a new outlook as the new season approaches.
        </p>
    <p><a href="">Read More</a></p>
  </div> <!-- mainfeature -->
  <h2>Recent Features</h2>
```

Now you can style the main feature area in your style sheet:

File: **styles.css (excerpt)**

```
#mainfeature {
  background-image: url(img/mainimg.jpg);
  background-repeat: no-repeat;
  background-color: #112236;
  color: white;
  padding: 2em 2em 1em 200px;
}
```

Here, we add the background image, `mainimg.jpg`, and set it to `no-repeat`. But if a user has the browser open to dimensions that are wider than the image, we don't want the exposed areas of the page to display white. To prevent this from happening, we add a background color of `#112236`; this is the same color as the far right-hand side of the image, so the image should appear to fade into the background color seamlessly. We then set the text color to `white` and use padding to position the text two ems from the top of the box, two ems from the right, one em from the bottom, and 200 pixels from the left-hand side, so that it's clear of the image of the footbag player.

Next, we style the heading and the paragraphs within the boxout:

File: **styles.css (excerpt)**

```
#mainfeature h2 {
  margin: 0;
  font-weight: normal;
  font-size: 140%;
}
#mainfeature p {
  font-size: 110%;
}
```

Finally, we need to style the "Read More" link that leads readers to the full article. Let's start by adding a `class="more"` attribute to the paragraph element so that we can target it with our style rules:

File: **index.html (excerpt)**

```
<div id="mainfeature">
  <h2>Simon Says</h2>
  <p>Simon Mackie tells us how a change of shoes has given him new
      moves and a new outlook as the new season approaches.</p>
  <p class="more"><a href="">Read More</a></p>
</div>
```


Order the print version of this book to get all 500+ pages!

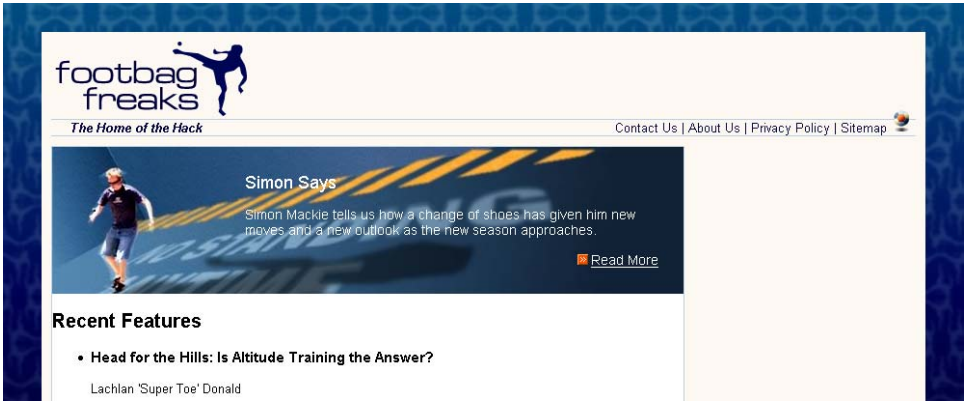
First, we remove the top margin from the paragraph that contains the link, to decrease the space between it and the paragraph. Then, we set `text-align` to `right`:

File: **styles.css (excerpt)**

```
#mainfeature p.more {
  margin-top: 0;
  text-align: right;
}
#mainfeature p.more a:link, #mainfeature p.more a:visited {
  color: white;
  background-image: url(img/more-bullet.gif);
  background-repeat: no-repeat;
  background-position: center left;
  padding-left: 14px;
}
```

We then style the `link` and `visited` pseudo-classes, changing their color to `white` and adding the `more-bullet.gif` background image. We only want to see the bullet once, so we set `repeat` to `no-repeat`, then position the background `center` and `left`. This positions the image in the center of the link's text. Finally, in order to stop the text from displaying over the top of the background image, we set `padding-left` to 14 pixels. The impact of these changes is shown in Figure 8.29.

Figure 8.29. After styling the main feature section



The Features List

Our layout is really starting to take shape now! Let's spend some time styling the main content on this page: the list of feature articles.

At the moment, the text inside the content area butts up against the border of the box. I want to create some space between that border and the content. The contents of the home page `content` `div` are enclosed in an unordered list, so one option we have is to add a margin to that list and to the `h2` above it. However, another page might have a different kind of main content, so in order that all of the pages can be dealt with in the same way, let's add another `div`, which wraps around the heading and features list, and give it a `class` of `inner`:

File: **`index.html` (excerpt)**

```
<div id="content">
  <div id="mainfeature">
    <h2>Simon Says</h2>
    <p>Simon Mackie tells us how a change of shoes has given him
        new moves and a new outlook as the new season approaches.
        </p>
    <p class="more"><a href="">Read More</a></p>
  </div> <!-- mainfeature -->
  <div class="inner">
    <h2>Recent Features</h2>
    <ul>
      <li>
        <h3>Head for the Hills: Is Altitude Training the
            Answer?</h3>
        <p>Lachlan 'Super Toe' Donald</p>
       <p>Vestibulum ante ipsum primis in faucibus orci luctus et
            ultrices posuere cubilia Curae; Praesent hendrerit
            iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>Hack up the Place: Freestylin' Super Tips</h3>
        <p>Jules 'Pony King' Szemere</p>
       <p>Vestibulum ante ipsum primis in faucibus orci luctus et
            ultrices posuere cubilia Curae; Praesent hendrerit
            iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>The Complete Black Hat Hacker's Survival Guide</h3>
        <p>Mark 'Steel Tip' Harbottle</p>
```


Order the print version of this book to get all 500+ pages!

```
      <p>Vestibulum ante ipsum primis in faucibus orci luctus et
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
      <li>
        <h3>Five Tricks You Didn't Even Know You Knew</h3>
        <p>Simon 'Mack Daddy' Mackie</p>
       <p>Vestibulum ante ipsum primis in faucibus orci luctus et
          ultrices posuere cubilia Curae; Praesent hendrerit
          iaculis arcu.</p>
        <p><a href="">Full Story</a></p>
      </li>
    </ul>
  </div>
</div> <!-- content -->
```

To create some space between the features list and the border of the containing box, let's add a margin to `#content .inner` in the style sheet:

File: **styles.css (excerpt)**

```
#content .inner {
  margin: 10px 20px 10px 40px;
}
```

If you view your layout in the browser, you should see the space that this margin creates. We can now address the content of this section.

First, let's style the heading. In our layout image, the heading has a blue underline that stretches across the entire width of the content—an effect we can create using a bottom border. Let's also add a small amount of padding to the bottom of the `h2`, to insert some space between the text and this border:

File: **styles.css (excerpt)**

```
#content .inner h2 {
  color: #245185;
  padding-bottom: 0.2em;
  border-bottom: 1px solid #b9d2e3;
  font-size: 110%;
}
```

Next, let's add a rule to remove the margin and list bullets from the list of feature items. While we could simply create this rule for `#content .inner ul`, as there's only one list in this page's layout, that approach might cause problems on other pages whose content includes lists that are not like this special features list. So

let's add a `class="features"` attribute to the `ul` element first, so we can style this particular list—and others like it—without affecting any normal, non-feature lists within page content:

File: **index.html (excerpt)**

```
<div class="inner">
  <h2>Recent Features</h2>
  <ul class="features">
    <li>
```

File: **styles.css (excerpt)**

```
#content .inner ul.features {
  margin: 0;
  padding: 0;
  list-style: none;
}
```

Each feature has a level three heading; we'll style these by increasing the font size:

File: **styles.css (excerpt)**

```
#content .inner h3 {
  font-size: 130%;
}
```

Let's also make each of these headings act as a link to the appropriate article on the Footbag Freaks site. We can style the `link` and `visited` pseudo-classes, as well:

File: **index.html (excerpt)**

```
<li>
  <h3><a href="">Head for the Hills: Is Altitude Training the
      Answer?</a></h3>
  <p>Lachlan 'Super Toe' Donald</p>
  <p>Vestibulum ante ipsum primis in faucibus orci luctus et
      ultrices posuere cubilia Curae; Praesent hendrerit iaculis
      arcu.</p>
  <p><a href="">Full Story</a></p>
</li>
<li>
  <h3><a href="">Hack up the Place: Freestylin' Super Tips</a></h3>
  <p>Jules 'Pony King' Szemere</p>
  <p>Vestibulum ante ipsum primis in faucibus orci luctus et
      ultrices posuere cubilia Curae; Praesent hendrerit iaculis
      arcu.</p>
```


Order the print version of this book to get all 500+ pages!

```
  <p><a href="">Full Story</a></p>
</li>
```

Example 8.52. `styles.css` (excerpt)

```
#content .inner h3 a:link, #content .inner h3 a:visited {
  color: #245185;
}
```

Finally, let's style the page's paragraph text by making it a dark gray and decreasing the font size to 90%:

File: **styles.css (excerpt)**

```
#content .inner p {
  color: #666666;
  font-size: 90%;
}
```

The Author Images

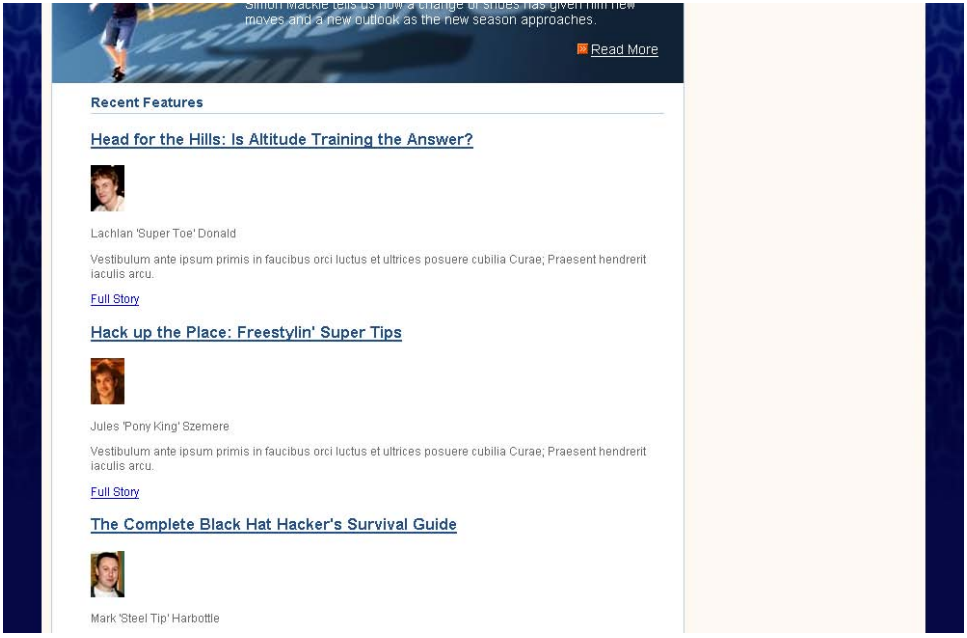
We want to display an image of the author alongside each feature article listing. Add the image to each feature item, after the heading, like so:

File: **styles.css (excerpt)**

```
<li>
  <h3><a href="">Head for the Hills: Is Altitude Training the
      Answer?</a></h3>
  <img src="img/lachlan.jpg" alt="Lachlan Donald" height="48"
      width="35" />
  <p>Lachlan 'Super Toe' Donald</p>
  <p>Vestibulum ante ipsum primis in faucibus orci luctus et
      ultrices posuere cubilia Curae; Praesent hendrerit iaculis
      arcu.</p>
  <p class="more"><a href="">Full Story</a></p>
</li>
```

This markup produces the display shown in Figure 8.30.

Figure 8.30. Displaying author images in the document



Let's use the `float: left` declaration to move these author shots to the left of the paragraph text. Note that we don't need to include the image's width here, as each `img` already has a width defined.

File: **styles.css (excerpt)**

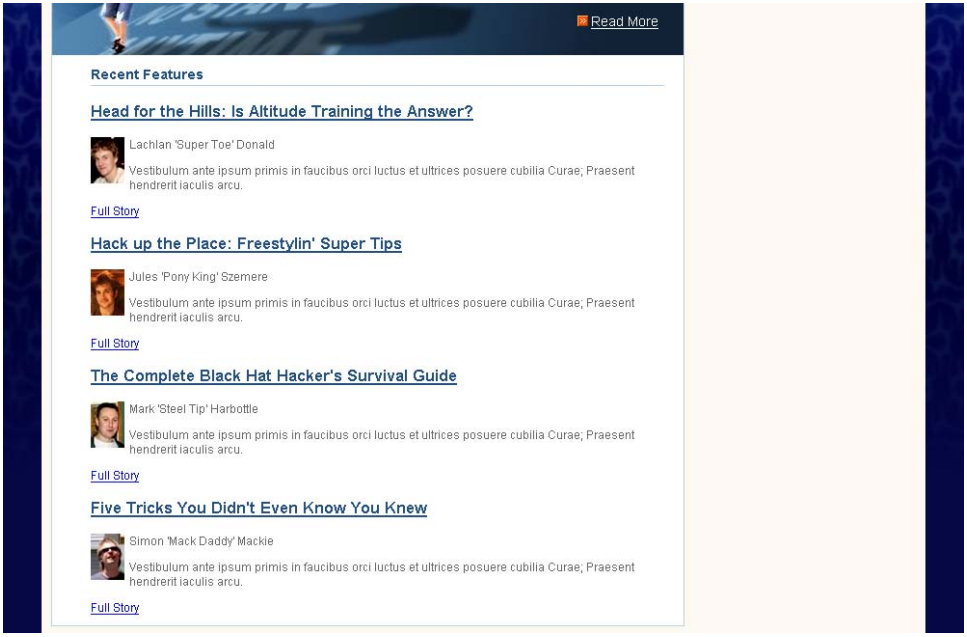
```
#content .inner .features li img {
  float: left;
  margin: 0 5px 5px 0;
}
```

Here, we've used a selector that will address only those images that are within an `li` element with the `class="features"` attribute. This way, we avoid affecting any other images that might be added to your content.

We've set the image to float left, and added a margin so that the text doesn't sit right next to the image—it has a little breathing room, as Figure 8.31 shows.

Order the print version of this book to get all 500+ pages!

Figure 8.31. Floating the author image



In our layout graphic, author names appear in bold text, so let's give the paragraph surrounding the author name a `class` attribute with the value `author`, and use a CSS rule to style it bold. We're not doing this with any `<strong>` or `<b>` tags because we're styling the author names purely for aesthetic reasons—not for any structural purpose. By keeping the author name styles out of the page markup, we're sticking to our goal of separating content from presentation. And, since we're using CSS, if we want to change the way the author name displays in future, we can simply edit the rules for the appropriate class, instead of finding every page on which an author's name is displayed and changing it there. Here's the change we need to make to the page markup, followed by the CSS rule that will make all suitably marked-up author names bold:

File: **`index.html` (excerpt)**

```
<img src="img/lachlan.jpg" alt="Lachlan Donald" height="48"
    width="35" />
<p class="author">Lachlan 'Super Toe' Donald</p>
<p>Vestibulum ante ipsum primis in faucibus orci luctus et ultrices
    posuere cubilia Curae; Praesent hendrerit iaculis arcu.</p>
```

File: **styles.css (excerpt)**

```
#content .inner p.author {
  font-weight:  bold;
}
```

The final page element that we need to style for this section is the "Full Story" links that appear beneath each feature. Add a `class` of `more` to each link's opening `<p>` tag, then add the following rules to your style sheet:

File: **styles.css (excerpt)**

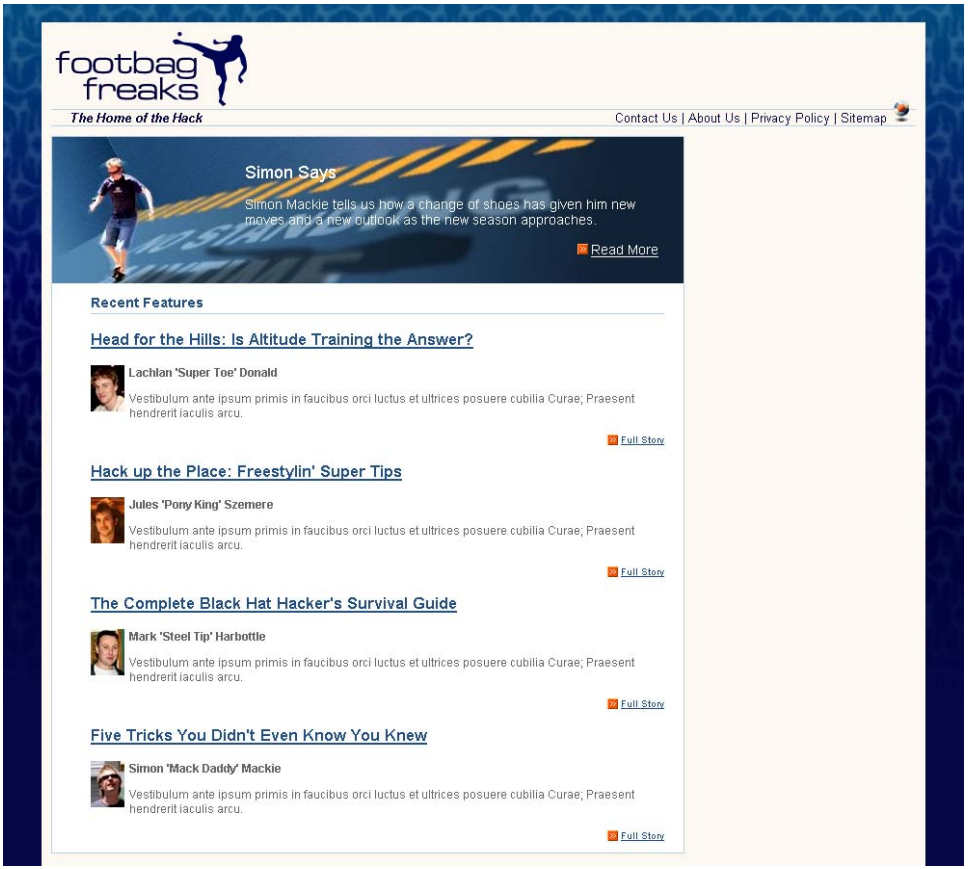
```
#content .inner p.more{
  margin-top: 0;
  text-align: right;
}
#content .inner p.more a:link, #content .inner p.more a:visited {
  color: black;
  background-image: url(img/more-bullet.gif);
  background-repeat: no-repeat;
  background-position: center left;
  padding-left: 14px;
  font-size: 90%;
  color: #1e4c82;
}
```

As I'm sure you've noticed, this styling is very similar to that of the "Read More" link within the feature article section at the top of the page.

Your layout should now look a lot like the original layout graphic. Our progress is shown in Figure 8.32. The page is very close to completion: we have only the sidebar left to style!

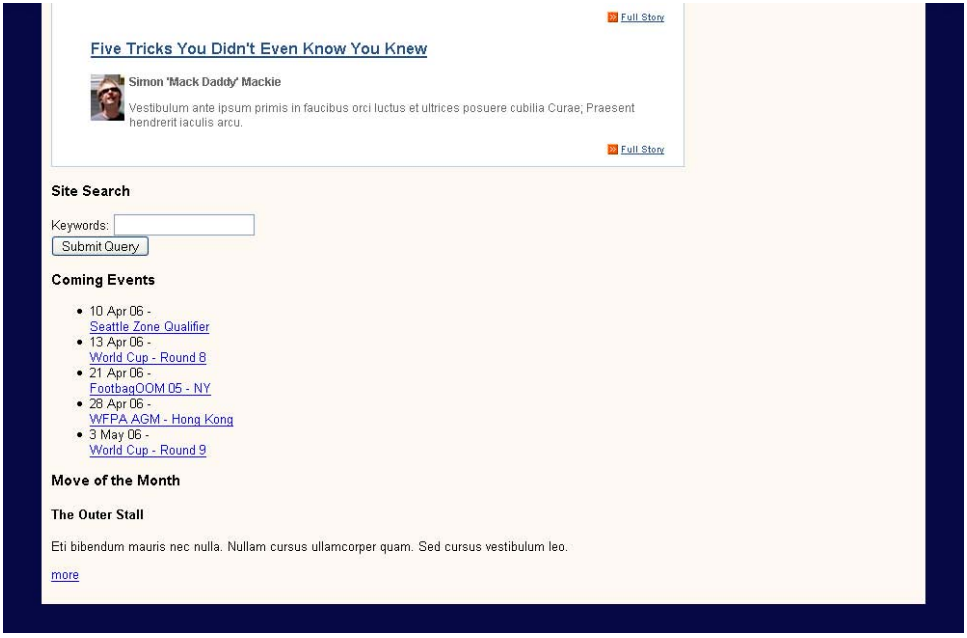
Order the print version of this book to get all 500+ pages!

Figure 8.32. Displaying the page after styling the main content area



The Sidebar

Figure 8.33. The unstyled sidebar



The content of the sidebar is languishing beneath the main content area, as Figure 8.33 illustrates. No rules have been applied to it, so it's just sitting in its natural location in the document.

Our first job is to move the sidebar from this position to the space we've created for it on the right of the content area.

First, let's see what happens if we position the sidebar using absolute positioning from the top and right. Add the following rules to your style sheet:

File: **styles.css (excerpt)**

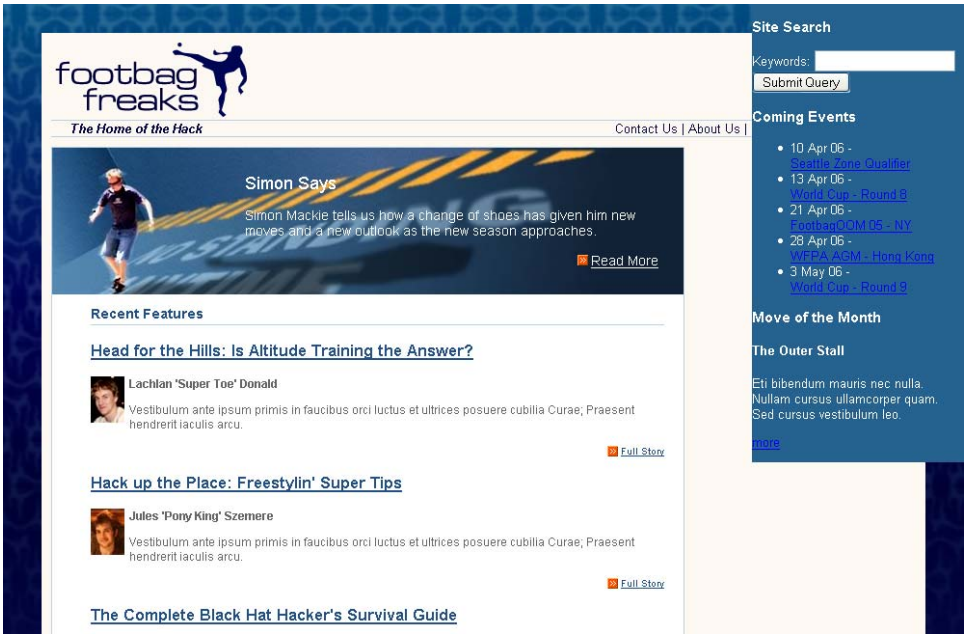
```
#sidebar {
  position: absolute;
  top: 0;
  right: 0;
  width: 220px;
  background-color: #256290;
```


Order the print version of this book to get all 500+ pages!

```
  color: white;
  margin: 0;
  padding: 0;
}
```

View your page in the browser. The sidebar is stuck to the top, right corner of the viewport as in Figure 8.34.

Figure 8.34. Positioning the sidebar top and right



When we discussed absolute and relative positioning earlier, I explained that an element is always positioned relative to its parent element's position, and that this concept was described as an element's positioning context. In this case, `#sidebar` doesn't have a positioned parent element, so it takes the viewport as its positioning context.

However, we do have an element that can be positioned to provide us with a useful positioning context—the `div` with `id="main"`.

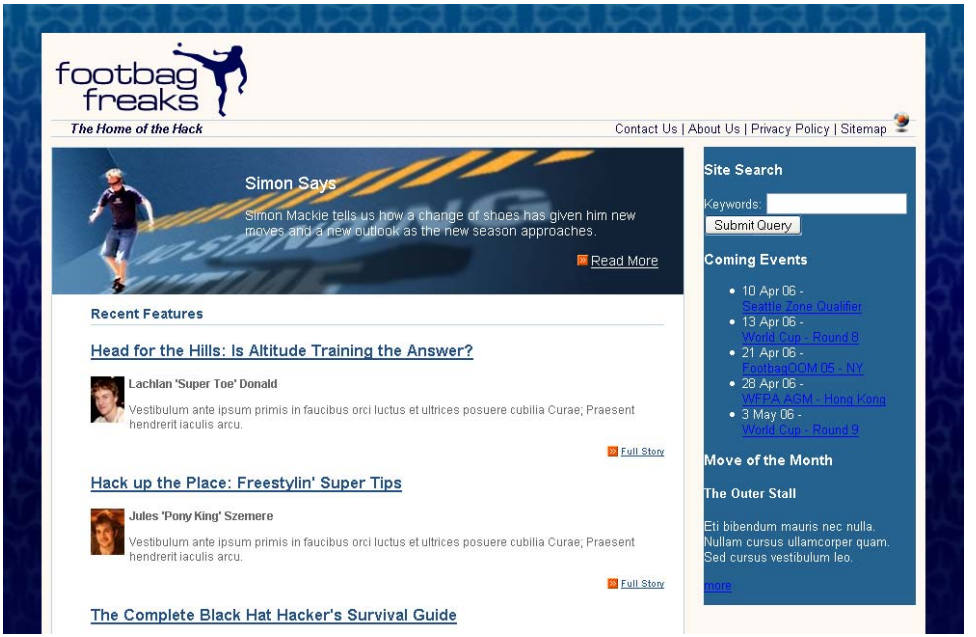
Find `#main` in your style sheet and add the following declarations:

File: **styles.css (excerpt)**

```
#main {
  position: relative;
  top:0;
  left:  0;
  width: 100%;
  margin-top: 10px;
}
```

The sidebar now takes `#main` as its parent, so it falls into place within the area defined by the `div` with that `id`, as Figure 8.35 illustrates.

Figure 8.35. Positioning the sidebar to the top and right of a relatively positioned container



With our sidebar now in position, we can start to style its contents. To start, we'll style the `h3` headings that head the different sections of the sidebar:

File: **styles.css (excerpt)**

```
#sidebar h3 {
  font-size: 110%;
  background-image: url(img/sidebar-header-bg.jpg);
```


Order the print version of this book to get all 500+ pages!

```
  background-repeat: no-repeat;
  margin: 0;
  padding: 0.2em 0 0.2em 10px;
  font-weight: normal;
}
```

Here, we're displaying a background image behind the heading to create the gradient effect we saw in our design comp.

Good Looks in the Background

Using a background image behind a heading is a great way to make your headings more attractive without resorting to using an image for the actual heading text. Using an image to display headings makes your site more difficult to maintain, as you need to manipulate those images every time you want to make even minor changes.

Let's have a closer look at the sections of content that display below each of the headings in the sidebar. We need to add a `div` with a `class` of `inner` to each of these, in order to create a little space and move the text content away from the border. Add this `div` to each of the three sections, as shown here:

File: **index.html (excerpt)**

```
<div id="sidebar">
  <div class="inner">
    <h3>Site Search</h3>
    <form method="post" action="" id="searchform">
      <div>
        <label for="keywords">Keywords</label>:
        <input type="text" name="keywords" id="keywords" />
      </div>
      <div>
        <input type="submit" name="btnSearch" id="btnSearch" />
      </div>
    </form>
  </div>
  <div class="inner">
    <h3>Coming Events</h3>
    <ul>
      <li>10 Apr 06 -<br /><a href="">Seattle Zone
          Qualifier</a></li>
      <li>13 Apr 06 -<br /><a href="">World Cup - Round 8</a></li>
      <li>21 Apr 06 -<br /><a href="">FootbagOOM 05 - NY</a></li>
      <li>28 Apr 06 -<br /><a href="">WFPA AGM - Hong
          Kong</a></li>
```

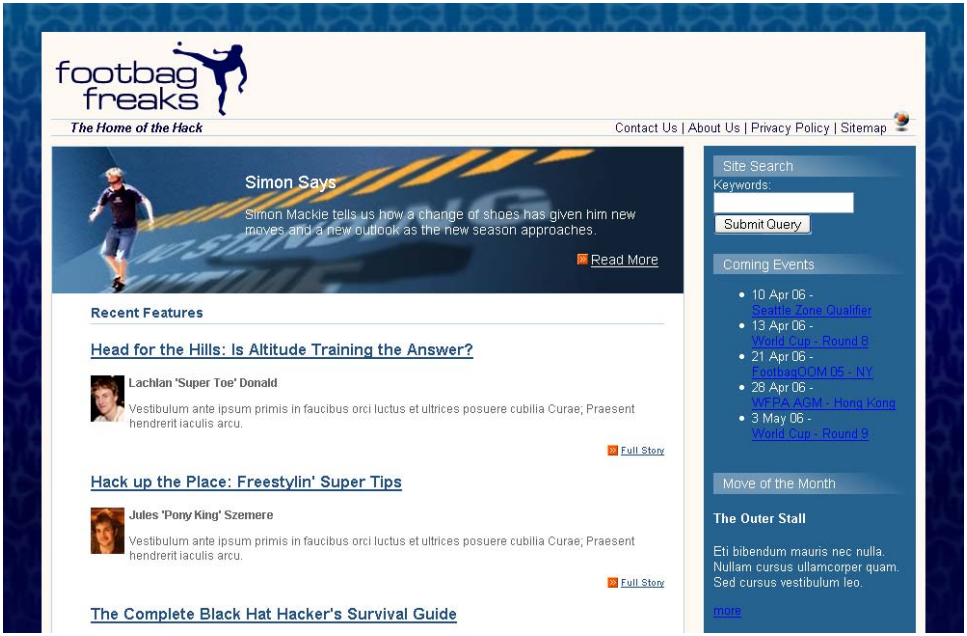
```
      <li>3 May 06 -<br /><a href="">World Cup - Round 9</a></li>
    </ul>
  </div>
  <div class="inner">
    <h3>Move of the Month</h3>
    <h4>The Outer Stall</h4>
    <p>Eti bibendum mauris nec nulla. Nullam cursus ullamcorper
        quam. Sed cursus vestibulum leo.</p>
    <p><a href="">more</a></p>
  </div>
</div> <!-- sidebar -->
```

Now, let's add ten pixels of padding to `inner`:

File: **styles.css (excerpt)**

```
#sidebar .inner {
  padding: 10px;
}
```

Figure 8.36. The display after styling the headings and `inner` class



Order the print version of this book to get all 500+ pages!

As you can see in Figure 8.36, the sidebar is starting to take shape. Now, let's address the list of events.

File: **styles.css (excerpt)**

```
#sidebar ul {
  list-style-image: url(img/more-bullet.gif);
  margin-left: 0;
  padding-left: 20px;
}
```

In the markup above, we use the `more-bullet.gif` image as the list bullet, remove the margin, and add left padding of 20 pixels in order to display the list in line with the headings.

File: **styles.css (excerpt)**

```
#sidebar p, #sidebar li {
  font-size: 90%;
  line-height: 1.4em;
}
```

Next up, we decrease the font size of the paragraph and list item text by reducing it to 90%. We also create a little more spacing between the lines with the help of the `line-height` property.

File: **styles.css (excerpt)**

```
#sidebar ul a:link, #sidebar ul a:visited {
  color:  white;
}
```

The links in the sidebar are white and underlined in the mock-up, so we set them to white with the rule above.

Finally, let's make all the dates in the event list bold. Add a `span` with `class="date"` to each of the dates in the list, then style them using the selector `#sidebar .date`, like this:

File: **index.html (excerpt)**

```
<ul>
  <li><span class="date">10 Apr 06</span> -<br />
      <a href="">Seattle Zone Qualifier</a></li>
  <li><span class="date">13 Apr 06</span> -<br /><a href="">World
      Cup - Round 8</a></li>
  <li><span class="date">21 Apr 06</span> -<br />
      <a href="">FootbagOOM 05 - NY</a></li>
```

```
  <li><span class="date">28 Apr 06</span> -<br /><a href="">WFPA
      AGM - Hong Kong</a></li>
  <li><span class="date">3 May 06</span> -<br /><a href="">World
      Cup - Round 9</a></li>
</ul>
```

File: **styles.css (excerpt)**

```
#sidebar .date {
  font-weight: bold;
}
```

The events in the sidebar now display as shown in Figure 8.37.

Figure 8.37. Displaying the styled events in the sidebar

The Form

It's time to create some rules for the search form at the top of the sidebar. Add `class="text"` to the `input type="text"` element, then create a rule for `#searchform .text` that gives the text box a width of 196 pixels and a border. Here's the markup:

File: **styles.css (excerpt)**

```
#searchform .text {
  width: 196px;
  border: 1px solid #45bac0;
}
```

Apply the `searchbutton` class to the `div` that surrounds the submit button, and add a rule for it to `styles.css`, setting `text-align` to `right` and adding a top margin so the button doesn't bump right up against the text box.

Order the print version of this book to get all 500+ pages!

File: **styles.css (excerpt)**

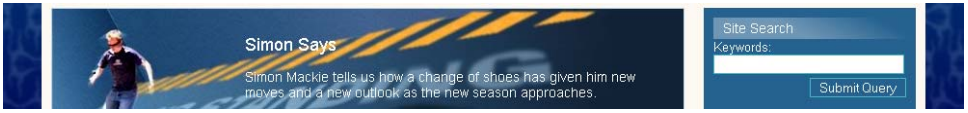
```
#searchform .searchbutton {
  text-align: right;
  margin-top: 4px;
}
```

Finally, let's style the button itself, giving it a border the same color as the text field, a background color that matches the blue used for the background of the sidebar, and a text color of white, as defined in the rules below. You'll also need to add a `class` attribute with the value `btn` to the `input` element. The results of your work should look like Figure 8.38.

File: **styles.css (excerpt)**

```
#searchform .btn {
  border: 1px solid #45bac0;
  background-color: #256290;
  color:  white;
}
```

Figure 8.38. Displaying the styled site search



Move of the Month

The final element of the sidebar that we need to consider is the Move of the Month section at its bottom. This section includes an image; we need to add this to the document first. Insert it below the `h4` and give it a `class` of `motm-image`:

File: **index.html (excerpt)**

```
<h3>Move of the Month</h3>
<h4>The Outer Stall</h4>
<img src="img/sidebar-player.gif"
    alt="player demonstrating the outer stall move" height="110"
    width="60" class="motm-image" />
<p>Eti bibendum mauris nec nulla. Nullam cursus ullamcorper quam.
    Sed cursus vestibulum leo.</p>
<p><a href="">more</a></p>
```

Let's float this image to the right so that we can display the text to one side of the image:

File: **styles.css (excerpt)**

```
#sidebar .motm-image {
  float: right;
  margin: 0 30px 0 20px;
}
```

As you can see, we've also added left and right margins to the image. The very last thing we need to do is to format the "more" link, which is very similar to the "Read More" and "Full Story" links in the rest of the layout. However, unlike those links, this link will normally appear next to a floated image. We want to ensure that it doesn't appear alongside the image: we want it always to display below. So, as you can see in the markup below, we use the `clear: right` declaration to ensure there are no floated elements to the right of the image. We'll also need to add the `more` class to the paragraph that contains the link:

File: **styles.css (excerpt)**

```
#sidebar p.more {
  clear: right;
  margin: 0 30px 0 0;
  text-align: right;
}
```

We'll be looking at `clear` in more detail in the next chapter. For now, note that it can take the values of `left` (clearing a left float), `right` (clearing a right float), and `both` (clearing both left and right floats). If you're using floated elements in your layouts, you'll find this a useful property.

The final rules, below, should be familiar to you from the other "Read More" and "Full Story" links:

File: **styles.css (excerpt)**

```
#sidebar p.more a:link, #sidebar p.more a:visited {
  color: white;
  background-image: url(img/more-bullet.gif);
  background-repeat: no-repeat;
  background-position: center left;
  padding-left: 14px;
}
```

This markup completes your two-column layout! The finished page display is shown in Figure 8.39.

Order the print version of this book to get all 500+ pages!

Figure 8.39. The completed two-column layout

Repositioning the Sidebar

We can really start to appreciate the flexibility of CSS layouts when we decide to experiment! For instance, imagine that we want to see how this layout would look if we positioned the sidebar on the left, rather than the right. To do this, you'd need to make only two changes in your CSS.

First, locate the `#content` rule and change the values for `margin`: give it a 240-pixel *left* margin, rather than a 240-pixel right margin. Then, set the right margin to `0`:

```
#content {
  margin: 0 240px 0 0;
  border: 1px solid #b9d2e3;
  background-color: white;
  color: black;
}
```

Now, find the `#sidebar` rule and change the positioning declaration `right: 0` to `left: 0`:

File: **styles.css (excerpt)**

```
#sidebar {
  position: absolute;
  top: 0;
  left: 0;
  width: 220px;
  background-color: #256290;
  color: white;
  margin: 0;
  padding: 0;
}
```

Save your style sheet and refresh the page in your browser. The sidebar will now appear on the left-hand side of the content, as Figure 8.40 shows.

Summary

We've covered a lot in this chapter! We began with an unstyled XHTML document, and after learning a little bit about the theory of using CSS for layout—in particular, absolute and relative positioning, margins, padding, and borders—we began to create a two-column layout using an absolutely positioned sidebar.

You now have a complete page layout that uses CSS positioning; it's the basic layout used by many of the sites we see on the Web every day. This layout method does have its limitations, though—we'll discover those, and discuss some alternative layouts, in the next chapter. However, if you need a two-column layout, this structure is robust and can be used as the basis for countless attractive site designs.

Order the print version of this book to get all 500+ pages!

Figure 8.40. Repositioning the sidebar

What's Next?

If you've enjoyed these chapters from *HTML Utopia: Designing Without Tables Using CSS, 2nd Edition*, why not order yourself a copy?

HTML Utopia: Designing Without Tables Using CSS, 2nd Edition is the definitive beginner's guide to CSS. This completely revised second edition is a step-by-step, clearly written tutorial that will make building beautiful, accessible, and maintainable CSS-based web sites a snap.

As standards guru Jeffrey Zeldman says, "After reading *HTML Utopia: Designing Without Tables Using CSS* you will not only understand how to use CSS to emulate old-school, table-driven web layouts, you will be creating Web sites that would be impossible to design using traditional methods"

HTML Utopia: Designing Without Tables Using CSS, 2nd Edition also includes download access to all of the best practice code samples and complete CSS layouts used throughout the book – plug them right into your own projects without any retyping!

In the rest of the book, you'll:

- Validate your CSS and make sure that it's correct.
- Learn to specify colors and fonts.
- Understand the cascade.
- Use CSS to position the elements on your page.
- Build robust, flexible two- and three-column layouts.
- Construct accessible forms using CSS.
- Learn to apply with the box model.
- Understand the differences between absolute and relative positioning.
- Build both fixed-width and liquid layouts.
- And a whole lot more…

On top of that, order direct from sitepoint.com and you'll receive a free 17" x 24" poster of your choice!

Order now and get it delivered to your doorstep!

Index

Symbols

A

B

Order the print version of this book to get all 500+ pages!

C

D

E

Order the print version of this book to get all 500+ pages!

F

Order the print version of this book to get all 500+ pages!

J

L

M

N

Order the print version of this book to get all 500+ pages!

Q

R

S

Order the print version of this book to get all 500+ pages!

T

U

V

W

X

Order the print version of this book to get all 500+ pages!

Z

Order the print version of this book to get all 500+ pages!