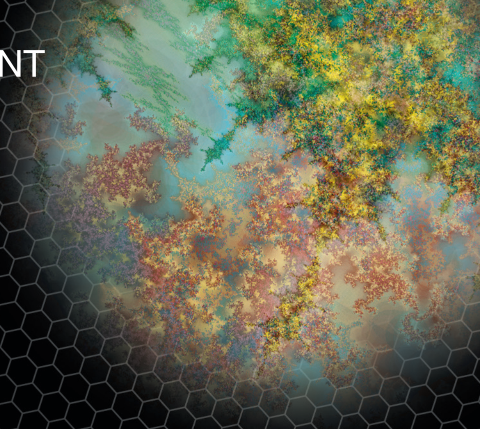
THE EXPERT'S VOICE® IN WEB DEVELOPMENT



Pro CSS and HTML Design Patterns

Increase creativity and productivity by using patterns in your web designs while leveraging CSS and (X)HTML best practices

Michael Bowers

Apress®

Pro CSS and HTML Design Patterns

Michael Bowers

Apress®

Pro CSS and HTML Design Patterns

Copyright © 2007 by Michael Bowers

ISBN-13 (pbk): 978-1-59059-804-7

ISBN-10 (pbk): 1-59059-804-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

I dedicate this book to my loving family.
To my wife, Teresa
To my son, Joshua
To my daughter, Sydney

They all sacrificed much to make this book possible.

Contents at a Glance

Contents

About the Author

MICHAEL BOWERS has been writing software professionally for 18 years. He taught himself to program when he was 14 and hasn't stopped since.

He has been the lead software developer, architect, and modeler for many projects ranging from web sites to application frameworks to database systems. He has built intranet applications, automated factories with robotics, developed languages along with interpreters and compilers, programmed handheld devices, integrated enterprise systems, and managed teams. His favorite languages include CSS, XHTML, XML, C#, C, Visual Basic, Java, JavaScript, ASP, and SQL.

Michael is also an accomplished pianist with a bachelor's degree in music composition, a master's degree in music theory, and an ABD PhD in music theory. In his spare time he loves to improvise, arrange, and compose music.

About the Technical Reviewer

■**PAUL HAINE** is a web designer currently working in London. He is the author of *HTML Mastery: Semantics, Standards, and Styling* (friends of ED, 2006) and runs a personal web site at `www.joeblade.com` alongside his design blog, `www.unfortunatelypaul.com`.

Acknowledgments

I want to thank my family, Teresa, Joshua, and Sydney, for the sacrifices they made so I could write this book. I especially want to thank my wife, Teresa. Without her continued support and encouragement, I could not have written this book. I also want to thank my mother and father for teaching me the values of service, hard work, and continuous improvement.

I want to thank the team at Apress for all their great work: Chris Mills as editor, Paul Haines as technical reviewer, Kylie Johnston as project manager, Ami Knox as copy editor, Laura Esterman as production editor, and all the others who have worked behind the scenes.

Introduction

This is a solutions book for styling HTML 4.01 and XHTML 1.1 with CSS 2.1. It contains more than 350 design patterns you can put to use right away. Each design pattern is modular and customizable, and you can combine patterns to create an unlimited number of designs.

Each design pattern has been thoroughly tested and proven to work in all major web browsers including Internet Explorer 7, Internet Explorer 6, Firefox 2, Opera 9, and Safari 2.

All the content in this book is usable and practical. You won't waste time reading about things that don't work! With this book, you will no longer have to use hacks, tricks, endless testing, and constant tweaking in multiple browsers to get something to work.

Using a design pattern is as easy as copying and pasting it into your code and tweaking a few values. This book shows you which values you can modify and how they affect the result so you can create the exact style and layout you want—without worrying whether it will work.

This is more than a cookbook. It systematically covers every usable feature of CSS and combines these features with HTML to create reusable patterns. Each pattern has an intuitive name to make it easy to find, remember, and talk about. Accessibility and best practices are carefully engineered into each design pattern, example, and source code.

You can read straight through the book, use it as a reference, and use it to find solutions. You'll love how the book's consistent layout makes it a joy to use.

The book puts examples on the left page and explanations on the right. Each example includes a screenshot and all relevant HTML and CSS code so you can easily see how each design pattern works. The explanation for each design pattern is included on the right-facing page of the example so you can easily study the example while you read how it works.

The layout also makes the book very usable as an e-book because you can see the example and explanation all on one screen; otherwise, you would have to flip back and forth between pages, which is difficult to do in an e-book.

Each design pattern has a name, which is placed at the top of each page. This makes it easy to find a design pattern, to remember it, and to talk about it with others. Since the name, screenshot, code, and explanation are placed in the same location in each set of facing pages, you can quickly thumb through the book to find what you are looking for.

Design patterns are organized by topic, and all usable CSS rules are covered in depth and in context like no other book. All design patterns are accessible and follow best practices, making this book a worthwhile read from cover to cover as well as an excellent reference to keep by your side while you are designing and coding.

This book unleashes your productivity and creativity in web design and development. Design patterns are like Legos—you can combine them in countless ways to create any design. They are like tools in a toolbox, and this book arms you with hundreds of tools you can whip out to solve problems quickly and reliably. Instead of hacking away at a solution, this book shows you how to create designs *predictably*—by combining *predictable patterns*.

Audience

This book is written for those who have some familiarity with CSS and HTML. It is for newcomers who have previously read an introductory book on CSS and HTML. It is for designers and developers who tried CSS at one time and gave up because it never seemed to work right. It is for professionals who want to take their CSS skills to a higher level. It is for all who want to create designs quickly without hacking around until they find something that works in all browsers.

This book assumes the reader knows the basics of *coding* CSS and HTML. If you work exclusively in WYSIWYG designers like Dreamweaver or FrontPage and never look at HTML or CSS code, you may find the code in this book overwhelming.

If you like to learn by example, like to see how code works, and have some familiarity with CSS and HTML, you will love this book.

In Chapters 17 and 20, seven design patterns use JavaScript. To fully understand them, you need to understand the basics of JavaScript, but you do not need to know JavaScript to use these patterns. Most importantly, you do not need to know anything about JavaScript to understand and use the remaining 343+ design patterns because they have nothing to do with JavaScript!

Innovations

This book contains several innovative concepts, terms, and approaches. These are not new or radical: the technology is already built into the major browsers, the concepts are implied in the CSS specification, and the terms are commonly used. What makes them innovative is how I define and use them to show what can be done with CSS and HTML. In other words, they are innovative because they simplify learning, understanding, and using CSS and HTML. These ideas change how you think about CSS and HTML, and that makes all the difference. Furthermore, many of the design patterns in the book are innovative because they document combinations of properties and elements to solve difficult problems like never before.

Six Box Models

One innovation in the book is the idea that CSS has *six* box models instead of one. CSS officially has one box model that defines a common set of properties and behaviors. A single box model is a very useful concept, but it is oversimplified. Over the years, I learned the hard way that box model properties work differently depending on the type of box.

This is one reason why so many people struggle with CSS. The box model seems simple, yet when one uses a box model property, such as `width`, it only works some of the time or may work differently than expected. For example, the `width` property sets the interior width of a block box, but on table boxes it sets the outer width of the border, and on inline boxes it does absolutely nothing.

Rather than treating different behaviors as an exception to one very complicated box model, I define six simple box models that specify the behavior for each type of box. Chapter 4 presents the six box models, which are inline, inline-block, block, table, absolute, and float. Since you always know which of these six box models you are using, you always know how each box model property will behave.

Furthermore, each box model defines its own way that it flows or is positioned. For example, inline boxes flow horizontally and wrap across lines. Block boxes flow vertically. Tables flow their cells in columns and rows. Floats flow horizontally, wrap below other floats, and push inline boxes and tables out of the way. Absolute and fixed boxes do not flow; instead, they are removed from the flow and are positioned relative to their closest positioned ancestor.

Box Model Extents

Another innovation in the book is the concept that there are three ways a box can be dimensioned: it can be sized, shrinkwrapped, or stretched (see Chapter 5). Each type of box requires different combinations of properties and property values for it to be sized, shrinkwrapped, or stretched. Various design patterns in Chapters 5 through 9 show how this is done. These three terms are not official CSS terms, but they are implied in the CSS 2.1 specification in its formulas and where it mentions "size," "shrink-to-fit," and "stretch."[1]

Of course, sizing, shrinkwrapping, and stretching are not new ideas. What is innovative is that this book clearly defines these three terms and shows how they are a foundational feature of CSS and a key *generator* of CSS design patterns.

Box Model Placement

Another innovation is the idea that there are three ways a box can be placed in relation to its container or its siblings: specifically, it can be indented (or outdented), offset from its siblings, or aligned and offset from its container (see Chapter 8). The CSS 2.1 specification talks much about *offsetting* positioned elements, and it talks a little about *aligning* elements (see Chapter 9 of the CSS 2.1 specification), but it does not discuss how elements can be *indented*, although this behavior is implied in its formulas.

Indenting, offsetting, and aligning are different behaviors. For example, an *indented* box is stretched and its margins shrink its width, whereas an *aligned* box is sized or shrinkwrapped and its margins do not shrink its width. Aligned and indented boxes are aligned to their containers, whereas offset boxes can be offset from their container or offset from their siblings.

Different combinations of properties and property values are needed to indent, offset, and align different types of boxes. The design patterns in Chapters 8 and 9 show how this is done.

Of course, indenting, offsetting, and aligning are not new ideas. What is innovative is that this book clearly defines these three terms and shows how they are a foundational feature of CSS and a key *generator* of CSS design patterns.

---

1. In the CSS 2.1 specification, the terms "size" and "sized" occur 15 times in Chapters 8, 9, 10, 11, 17, and 18. These occurances refer to the general sense that a box has size.

   The terms "shrink" and "shrink-to-fit" occur 9 times in Chapters 9 and 10 of the CSS 2.1 specification. The idea that different boxes can shrinkwrap to fit their content is implied in Sections 10.3.5 through 10.3.9 and Section 17.5.2.

   The terms "stretch" and "stretched" occur 4 times in Chapters 9 and 16. The idea of stretching a box to its container is mentioned in passing as shown in the following quote (italics added), "many box positions *and sizes* are calculated with respect to the edges of a rectangular box called a containing block." (See Sections 9.1.2, 9.3.1, and 10.1.)

Column Layouts

Another innovation is the discovery, naming, and documenting of 12 automated techniques built into browsers for laying out columns in tables (see Chapter 16).

All the major browsers include these powerful column layout features. They are compatible across the major browsers and are very reliable. Even though using tables for page layout is not recommended,[2] *tabular data* still needs to be laid out, and you can take advantage of these column layouts to make tabular data look great.

Fluid Layouts

Another innovation is Fluid Layouts (see Chapter 17). The concept of fluid layouts is not new, but the process of creating them is commonly one of trial and error. In Chapter 17, I present four simple design patterns you can use to create complex fluid layouts with confidence and predictability in all major browsers.

These design patterns, Outside-in Box, Floating Section, Float Divider, and Fluid Layout, use floats and percentage widths to make them fluid, but they do so without the problems you normally encounter using these techniques, such as collapsed containers, staggered floats, and percentages that push floats below each other.[3]

The Fluid Layout design pattern creates columnar layouts with the versatility of tables but without using tables. Even better than tables, these layouts automatically adjust their width and reflow from columns into rows as needed to fit into narrow displays.

Event Styling

Another innovation is the Event Styling JavaScript Framework presented in Chapter 17. This is a simple, powerful, open source framework for *dynamically and interactively* styling a document. It uses the latest best practices to ensure that HTML markup is completely free of JavaScript code and completely accessible, and all styling is done with CSS. Furthermore, the framework allows you to select elements in JavaScript using the *same selectors* you use to select elements in CSS. This vastly simplifies and unifies the styling and scripting of a dynamic HTML document!

The book includes this framework to show how to integrate JavaScript, CSS, and HTML so you can use styles interactively. Of course, if you do not want to use JavaScript, you can skip over the five JavaScript design patterns in Chapter 17 and the two JavaScript patterns in Chapter 20—the remaining 343+ design patterns do not use JavaScript.

Combining HTML and CSS to Create Design Patterns

The final and most pervasive innovation in the book is the idea of combining general *types* of HTML elements with CSS properties to create design patterns. The book defines four major

2. Using tables for layout creates accessibility issues for nonsighted users. Furthermore, fluid layout techniques (as shown in Chapter 17) are completely accessible and much more adaptable than tables.

3. Internet Explorer 6 has a number of *bugs* that may occur when you float elements. Unfortunately, there is no way to create a solution that always bypasses these bugs, although the Fluid Layout design pattern does a good job of avoiding them most of the time. Fortunately, Internet Explorer 7 fixes these bugs.

types of HTML elements in Chapter 2 (structural block, terminal block, multi-purpose block, and inline) and Chapter 4 maps them to the six box models (inline, inline-block, block, table, absolute, and float).

Each design pattern specifies how it applies to *types* of HTML elements. In other words, a design pattern is more than a recipe that works only when you use specific elements; it is a pattern that applies to all equivalent *types* of HTML elements.

For example, the Floating Drop Cap design pattern in Chapter 18 specifies a pattern that uses block and inline elements, but it does not specify which block and inline elements you have to use (see Listing 1). For example, you could use a paragraph for the `BLOCK` element and a span for the `INLINE` element (see Listing 2), or you could use a division for the `BLOCK` and a `<strong>` for the `INLINE`, and so forth.

In some exceptional cases, a design pattern may specify an actual element, like a `<span>`. This happens when a specific element is the best solution, the only solution, or an extremely common solution. Even in these cases, you can usually swap out the specified element for another element of the same type. You can even use a different type of element as long as it produces valid XHTML and you change its box model to be compatible (see the Display design pattern and the box models in Chapter 4; also see Blocked in Chapter 11, Inlined in Chapter 13, and Tabled, Rowed, and Celled in Chapter 15).

Listing 1. *Floating Drop Cap* ***Design Pattern***

HTML

```
<BLOCK class="hanging-indent">
  <INLINE class="hanging-dropcap"> text </INLINE>
</BLOCK>
```

CSS

```
*.hanging-indent { padding-left:+VALUE; text-indent:-VALUE; margin-top:±VALUE; }
*.hanging-dropcap { position:relative; top:±VALUE; left:-VALUE; font-size:+SIZE;
  line-height:+SIZE; }
```

Listing 2. *Floating Drop Cap* ***Example***

HTML

```
<p class="hanging-indent">
  <span class="hanging-dropcap" >H</span>anging Dropcap.
</p>
```

CSS

```
*.hanging-indent { padding-left:50px; text-indent:-50px; margin-top:-25px; }
*.hanging-dropcap { position:relative; top:0.55em; left:-3px; font-size:60px;
  line-height:60px; }
```

Conventions

Each design pattern uses the following conventions:

- Uppercase tokens should be replaced with actual values. (Notice how the uppercase tokens in Listing 1 are replaced with values in Listing 2.)
- **Elements** are uppercase when you should replace them with elements of your choice. If an element name is lowercase, it should not be changed unless you ensure the change produces the same box model. The following are typical element placeholders:
  - **`ELEMENT`** represents any type of element.
  - **`INLINE`** represents inline elements.
  - **`INLINE_TEXT`** represents inline elements that contain text such as `<span>`, `<em>`, or `<code>`.
  - **`BLOCK`** represents block elements.
  - **`TERMINAL_BLOCK`** represents terminal block elements.
  - **`INLINE_BLOCK`** represents inline block elements.
  - **`HEADING`** represents `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`.
  - **`PARENT`** represents any element that can be a valid parent of its children.
  - **`CHILD`** represents any element that can be a valid child of its parent.
  - **`LIST`** represents any list element including `<ol>`, `<ul>`, and `<dl>`.
  - **`LIST_ITEM`** represents any list item including `<li>`, `<dd>`, and `<dt>`.
- **Selectors** that you should replace are uppercase. If a selector contains lowercase text, that part of the selector should not be changed unless you also modify the HTML pattern, such as changing a class name. The following are typical placeholders:
  - **`SELECTOR {}`** represents any selector.
  - **`INLINE_SELECTOR {}`** represents any selector that selects inline elements.
  - **`INLINE_BLOCK_SELECTOR {}`** represents any selector that selects inline-block elements.
  - **`BLOCK_SELECTOR {}`** represents any selector that selects block elements.
  - **`TERMINAL_BLOCK_SELECTOR {}`** represents any selector that selects terminal block elements.
  - **`SIZED_BLOCK_SELECTOR {}`** represents any selector that selects sized block elements.
  - **`TABLE_SELECTOR {}`** represents any selector that selects table elements.
  - **`CELL_SELECTOR {}`** represents any selector that selects table cell elements.

- **`PARENT_SELECTOR {}`** represents any selector that selects the parent in the design pattern.
- **`SIBLING_SELECTOR {}`** represents any selector that selects the children in the pattern.
- **`TYPE {}`** represents a selector that selects elements by a type of your choice such as `h1` or `span`.
- **`*.CLASS {}`** represents a selector that selects elements by a class name of your choice.
- **`#ID {}`** represents a selector that selects elements by an ID of your choice.

- **Values** that you should replace are represented by uppercase tokens. If a value contains lowercase text, that part of the value should not be changed. The following are typical value tokens:
  - Some values are literal and not meant to be replaced such as `0`, `-9999px`, `1px`, `1em`, `none`, `absolute`, `relative`, and `auto`. These values are always lowercase.
  - **`+VALUE`** represents a positive measurement greater than or equal to zero, such as `0`, `10px`, or `2em`.
  - **`-VALUE`** represents a positive measurement less than or equal to zero, such as `0`, `-10px`, or `-2em`.
  - **`±VALUE`** represents any measurement.
  - **`VALUEem`** represents an em measurement.
  - **`VALUEpx`** represents a pixel measurement.
  - **`VALUE%`** represents a percentage measurement.
  - **`VALUE_OR_PERCENT`** represents a value that can be a measurement or a percentage.
  - **`WIDTH STYLE COLOR`** represents multiple property values, such as those required by `border`. I use an uppercase token for each value.
  - **`url("FILE.EXT")`** represents a background image where you replace `FILE.EXT` with the URL of the image.
  - **`CONSTANT`** represents a valid constant value. For example, `white-space` allows three constant values: `normal`, `pre`, and `nowrap`. For convenience, I often list the valid constant values in uppercase with underscores in between each possible value, such as `NORMAL_PRE_NOWRAP`.
  - **`ABSOLUTE_FIXED`** represents a list of constant values from which you can choose one value. The underscore separates the constant values. The complete list of values for `position` includes `static`, `relative`, `absolute`, and `fixed`. If a design pattern only works for `absolute` and `fixed`, the pattern specifies `position:ABSOLUTE_FIXED`. If it works for all four values, it specifies `position:STATIC_RELATIVE_ABSOLUTE_FIXED` or `position:CONSTANT`.

- **`-(TAB_BOTTOM + EXTRA_BORDER + EXTRA_PADDING)`** is an example of a formula that you would replace with a calculated value. The uppercase tokens in the formula are tokens that occur elsewhere in the design pattern. For example, if you assigned `TAB_BOTTOM` to `10px`, `EXTRA_BORDER` to `10px`, and `EXTRA_PADDING` to `10px`, you would replace the formula with `-30px`.

Using This Book

You can use the book to master CSS. You can read straight through the book to take your CSS skills to a higher level and to discover the many golden nuggets tucked away inside design patterns. Each chapter is organized so that it builds on design patterns presented earlier in the chapter and presented in previous chapters. On the other hand, since individual chapters and design patterns are self-contained, you can read them one by one in any sequence to master a specific topic or technique.

You can use the book as a reference book. This book explains all of the usable CSS properties and shows how to use them in examples. Even more importantly, many properties behave differently when combined with other properties. Each design pattern identifies and documents the unique combination of properties required to create a specific result. This makes it a reference book not only for how CSS properties work alone, but also for how they work *in combination*.

You can use the book to learn by example. Since all examples in the book follow best practices, you can learn good habits and techniques just by studying them. To make studying the book by example easier, you can use the "See also" sections to look up all related design patterns. This allows you to easily see many examples of how a specific CSS property or feature can be used in a variety of contexts.

You can use the book as a cookbook to help you create designs or to solve problems. Design patterns are organized by topic so you can quickly find related solutions.

I have added extra features to the book to make it easy to find a solution when you need it. You can use the table of contents, the index, thumb tabs, chapter outlines, design pattern names, and the "See also" section of each design pattern to quickly find properties, patterns, answers, and solutions. Since the screenshots in each example are in the same location on every page, you can even thumb through the book while looking at screenshots to find a solution. I find visual scanning a very easy, fast, and effective way to find solutions!

Companion Web Site

The companion web site, **cssDesignPatterns.com**, is designed to enhance your experience with the book. Each page contains links to related design patterns and a search box for finding patterns. Each design pattern contains the example and its source code as well as additional information, such as additional examples, errata, comments, and links to related resources on the Internet.

At the end of each design pattern in the book is a link to the design pattern on cssDesignPatterns.com. (Each design pattern on the web site is a directory named after the design pattern with spaces in the design pattern name replaced by hyphens.)

In addition, cssDesignPatterns.com contains design patterns that are not in the book.

How This Book Is Structured

Chapters 1 through 3 explore the fundamentals of CSS and HTML:

- **Chapter 1 shows how design patterns make CSS easy.** Here I demonstrate how to combine simple design patterns into more complex and powerful patterns. I also review the syntax of CSS and the cascade order. In addition, I present several charts that make using CSS easy: a list of links to useful CSS web sites, a one-page summary of CSS properties; a four-page listing of all usable CSS *properties, values, and selectors* organized by where they can be used; charts on units of measure and font size; two example stylesheets for normalizing the styles of elements in all browsers; and a 12-step guide to troubleshooting CSS.
- **Chapter 2 introduces the design patterns that underlie HTML.** In this chapter, I present the best practices of using HTML including coding in XHTML. I also explore the types of structures you can create with HTML including structural blocks, terminal blocks, multi-purpose blocks, and inlines. I also show how to use IDs and attributes for easy selection by CSS selectors.
- **Chapter 3 introduces design patterns for CSS selectors and inheritance.** Here I demonstrate how selectors are the bridge between HTML and CSS. I present design patterns for type, class, ID, position, group, attribute, pseudo-element, pseudo-class, and subclass selectors. I also explore CSS inheritance.

Chapters 4 through 6 explore the six CSS box models. They show how each HTML element is rendered as one of these six types of boxes (or not rendered at all). They demonstrate how the same properties produce different results in each box model, and how each box model flows differently from the other box models.

- **Chapter 4 explores the six box models**: inline, inline-block, block, table, absolute, and float.
- **Chapter 5 explores the three ways of dimensioning a box**: sized, shrinkwrapped, or stretched.
- **Chapter 6 explores each of the box model properties**: margin, border, padding, background, overflow, visibility, and pagebreak.

Chapters 7 through 9 explore how boxes flow or are positioned.

- **Chapter 7 explores the five positioning models** (static, absolute, relative, fixed, and floated) and relates them to the six box models.
- **Chapter 8 explores the three ways a box can be positioned**: for example, a box can be indented or outdented, offset from its siblings, or aligned and offset from its container.
- **Chapter 9 combines the patterns in Chapters 7 and 8**: The combinations result in more than 50 design patterns for positioning elements—with a particular focus on absolute and fixed positioning.

Chapters 10 through 12 explore in detail how inline boxes flow and how to style, space, and align text and objects:

- **Chapter 10 explores the properties that style text** and also contains three design patterns for hiding text while remaining accessible to nonsighted users.
- **Chapter 11 shows how to *space* inline content** horizontally and vertically.
- **Chapter 12 shows how to *align* inline content** horizontally and vertically.

Chapters 13 and 14 explore in detail how blocks and images flow and how they can be styled:

- **Chapter 13 explores blocks**, starting with a discussion of the structural meaning of blocks and how you can visually display that meaning. It covers lists, inlining blocks, collapsed margins, run-in blocks, block spacing, and marginal blocks.
- **Chapter 14 explores images**, such as image maps, semi-transparent images, replacing text with images, sprites, shadowed images, and rounded corners.

Chapters 15 and 16 explore in detail how to style and lay out tables and cells.

- **Chapter 15 explores tables** including table selectors, collapsed borders, hiding cells, vertically aligning content in cells, and displaying inline and block elements as tables.
- **Chapter 16 explores laying out table columns using 12 patterns**, which automatically shrinkwrap columns, size them, proportionally distribute them, and so forth.

Chapter 17 explores how the flow of floats can be used to create fluid layouts:

- **Chapter 17 shows how to create fluid layouts** that automatically adapt to different devices, fonts, widths, and zoom factors. It also shows how to create interactive layouts using JavaScript.

Chapters 18 through 20 show how to combine design patterns to create a variety of solutions to the same problem. Each solution addresses different needs and has different advantages and disadvantages. Besides being useful solutions in and of themselves, they demonstrate how you can combine patterns to solve any design problem.

- **Chapter 18 explores drop caps.** Here I cover seven types of drop caps using seven different combinations of design patterns.
- **Chapter 19 explores callouts and quotes.** The chapter demonstrates five types of callouts and three types of quotes.
- **Chapter 20 explores alerts.** Here I present three types of interactive alerts and eight types of text alerts (i.e., attention getters).

Downloading the Code

All code is available at `www.cssDesignPatterns.com`.

You can also download the code at `www.apress.com` by searching for and going to the detail page for *Pro CSS and HTML Design Patterns*. On the book's detail page is a link to the sample code compressed into a ZIP file. You can use a utility like WinZip to uncompress the code.

Using the Code

The code is arranged in folders, with a folder for each chapter. To make chapter folders easy to navigate, each folder name includes the chapter number and title. Inside each chapter folder are example folders: one for each design pattern presented in the chapter.

So you can easily find examples, each example folder has the same name as its design pattern. This makes it easy and fast to find design patterns by searching folder names. Since the HTML in each example names and describes its design pattern, you can find a design pattern by searching for words inside HTML files. You could also search inside CSS files for examples that use a particular CSS property, such as `display`.

To make it easy to view examples in multiple browsers, I put a file named `index.html` in the root folder that links to all design pattern folders. In turn, each folder contains a file named `index.html` that links to all the design patterns in that folder. These navigation pages make it quick to find and view each design pattern in each chapter.

Each example folder contains *all* the files needed to make the example work. This makes it a breeze to use the examples in your own work: simply copy a folder and start making changes. You don't have to worry about tracking down and including files from other folders.

The most important files in each example folder are `example.html` and `page.css`. `example.html` contains the XHTML code for the example. `page.css` is the main stylesheet for the example.

Each example also uses a CSS file named `site.css`. It contains a few nonessential font and heading rules that give all the examples in the book the same basic look and feel.

In a few exceptional cases, I use an additional CSS file to overcome bugs or nonstandard behavior in Internet Explorer.[4] `ie6.css` contains rules to fix problems in Internet Explorer 6. `ie7.css` contains rules to fix problems in Internet Explorer 7. `ie67.css` contains rules to fix problems in both versions 6 and 7. Rules in these files override rules in `page.css`.

The seven JavaScript examples use five JavaScript files. These are explained in the Event Styling design pattern Chapter 17. `page.js` is the most important file because it contains JavaScript code specific to the example. The remaining JavaScript files are open source libraries.

Lastly, each example folder contains all image files used by that example.

---

4. There are only 25 of these files out of more than 350 design patterns. Most of these files contain only a single, simple rule, such as `div{zoom:1;}`. In spite of the numerous bugs, quirks, and nonstandard features of Internet Explorer 6, I only needed to build workarounds into 25 design patterns. This is because I carefully designed the patterns in this book to avoid problems in the first place. I allowed an exception in a pattern only when I could find no alternative. I literally had to throw out hundreds of design patterns to find patterns that work without exception. Lastly, because Internet Explorer 7 fixes most of the bugs in Internet Explorer 6, only 4 of these 25 exceptions apply to Internet Explorer 7.

Errata

You can view errata at `www.cssDesignPatterns.com` and on the detail page of the book at `www.apress.com`.

If you find an error in the book, I would greatly appreciate knowing about it. Please e-mail the problem to `support@apress.com` and `support@cssDesignPatterns.com`.

Contacting the Author

You can contact me at `mike@cssDesignPatterns.com`. I look forward to your comments, suggestions, and questions.