Practical CSS3

DEVELOP AND DESIGN

Chris Mills

Practical CSS3

DEVELOP AND DESIGN

Chris Mills

Peachpit Press

Practical CSS3: Develop and Design
Chris Mills

Peachpit Press
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.peachpit.com
To report errors, please send a note to: errata@peachpit.com
Peachpit Press is a division of Pearson Education.

Acquisitions Editor: Rebecca Gulick
Development and Copy Editor: Anne Marie Walker
Technical Reviewers: Peter Gasston, Bruce Lawson
Production Coordinator: Myrna Vladic
Compositor: David Van Ness
Proofreader: Patricia Pane
Indexer: Valerie Haynes-Perry
Cover Design: Aren Howell Straiger
Interior Design: Mimi Heft

ISBN-13: 978-0-321-82372-4
ISBN-10: 0-321-82372-9

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

ACKNOWLEDGMENTS

I'd like to give a shout out to all the spiffing chaps and chapesses of awesomeness and beauty that have helped and inspired me during this time, and driven me to get this book written!

My colleagues and friends at Opera for being almost like a second family, for teaching me so much, for helping me fix my code, and for making web standards fun. ODevRel2012: Andreas, Bruce, Daniel-san, Karl, Luz, Mike, Patrick, Shwetank, Tiffany, Vadim, and Zi Bin. And thanks to all the other talented people who make Opera a great place to work.

My allies in the wider web dev community for giving me much inspiration and smiles, mainly on Twitter: Jake74, Dan Donald, Phil Sherry, Shaun/Leslie Jensen-Inman, Doug Schepers, Jon Hicks, Chris Murphy, and the rest of the Irish posse, Remy Sharp, Anna Debenham, Mark and Emma Boulton, and the rest of the FSS crew, Henny Swan, and the W3C Web Education Community Group—phew. If I forgot your name on this list, please abuse me on Twitter: @chrisdavidmills.

Peter Gasston for an awesome tech review job on this book. I owe you mate.

Anne Marie Walker, Rebecca Gulick, and the rest of the Peachpit crew for kicking my ass into delivering this thing and helping to shape it.

Conquest of Steel (Vic/DD/Dan/Claymore) for being almost like a third family, or maybe more like having four whinging girlfriends. Cheers guys for the 15 years and counting of heavy metal. \m/

My friends in other far-flung places for always giving me love and support, even if they didn't understand this interweb thing.

My parents for "bringing me up proper." I love you both very dearly.

And most of all I'd like to give thanks and love to Kirsty, Gabriel, Elva, and Freida for putting up with me for four months while ignoring them to write this book, and for being the main reason I get out of bed in the morning.

CONTENTS

ONLINE RESOURCES

Throughout this book I use several third-party, online resources that include scripts and stylesheets, and I present and reference many examples that I wrote to illustrate the concepts in this book. The third-party resources are referenced where appropriate, so you'll be able to find them when needed. To find my examples is even easier: You can download them all athttp://peachpit.com/practicalcss3.

But that's not all! Also available athttp://peachpit.com/practicalcss3 are the following:

- **A bonus chapter.** In Chapter 9, "Styling HTML5 Media and Forms," I discuss building custom-styled controls for your HTML5 `<video>` and `<audio>` elements, and styling form elements using the form-related pseudo-classes in CSS3.
- **A cheat sheet.** This reference document details the syntax of all the new CSS3 features I use in this book and how they are supported in browsers. Print it out and hang it on your wall as an at-a-glance guide! I'll update this reference as the data changes.

Both are courtesy of your very generous author.

WELCOME TO CSS3

CSS3 provides you with exciting new tools for your web development toolbox, allowing you to accomplish many styling tasks in a much easier, more flexible, and less hackish manner than you've been used to when working with CSS2. The following chapters will introduce you to the most useful, new CSS3 features and show you how to use them in real

HTML AND CSS BASIC KNOWLEDGE

This book assumes you are well versed in basic HTML(4) and CSS(2) features and techniques. But just in case you need to look up any of the basics, keep some decent reference material to hand. A wealth of excellent tutorials is available on the W3C Web Education Community Wiki at www.w3.org/community/webed/wiki/Main_Page.

THE LATEST, GREATEST BROWSERS

Be sure to install the latest versions of desktop Opera, Firefox, Chrome, Safari, and Internet Explorer (IE). Ideally, you should have a testing environment available for all modern browsers; have as many to hand as you can.

OLDER, LESS-CAPABLE BROWSERS

Have older, less-capable browsers available for testing fallbacks, polyfills, and graceful degradation. Run older versions of IE on multiple virtual machines (VirtualBox is an acceptable, free option at www.virtualbox.org). Camino is a good option for a test Mac-based browser that doesn't support most of the new CSS3/HTML5 features.

projects today, as well as provide alternatives and fallbacks for less-capable browsers. Before you start this book, make sure you have the following pre-requisites. Now that you have all of the tools you need laid out in front of you, you're ready to go and make beautiful CSS3 music. Let's get going.

ALTERNATIVE BROWSING DEVICES

To test sites on different screen sizes, resolutions, and control mechanisms, have at least one or two alternative browsing devices. Mobile phones and tablets are essential fodder. A web-enabled TV would also be fun!

DEBUGGING ENVIRONMENTS

When it comes to choosing debugging environments, you have so many choices! Dragonfly on Opera, Firebug on Firefox; hell, every browser tends to come with a respectable debugging environment these days. Be sure to become familiar with as many as possible so you'll have the best chance at tracking down irksome bugs.

A DECENT TEXT EDITOR

A good text editor is all you need to write CSS and HTML. Coda on the Mac is awesome (http://panic.com/coda), but it's not free. Good free alternatives are Notepad++ for Windows, Text Wrangler for Mac, and Bluefish for Linux. WYSIWYG environments are not recommended, especially for learning. I'm a big fan of Jared Spool's quip about them being more like "WYSI … WTF"!

1

INTRODUCTION TO CSS3 AND MODERN WEB DESIGN

CSS3, the new, modular version of the CSS3 spec, contains many awesome new features that will make your web design work easier, more flexible, and more interesting. What's not to love? Browser support is not complete yet, but many of the features have enough support to be useful in a production environment, and you can work around nonsupporting browsers.

In this chapter I'll provide the rationale behind why the new version came about and gently preach a manifesto of modern web design to you. Then I'll provide a brief roundup of the CSS3 modules before examining some of the general new features of CSS3 that are useful to explore as background knowledge before you go any further.

WHY **CSS3?**

CSS3 has been around for longer than you might think. In fact, work had started on the earliest parts of CSS3 at about the same time as the CSS2 spec was being finished in the late 1990s. CSS2 has many very powerful features, and you can do a lot with it, but it was clear all those years ago that despite this a number of features were missing from the spec. This was evidenced by the fact that web designers tried to do many tasks using weird and interesting hacks or unusual techniques, often involving lots of nested `<div>`s or other semantic backstabbery, images, or even proprietary technologies like Flash. Some examples that spring to mind include:

- Font embedding. Downloading custom fonts for use on websites has been available in Internet Explorer (IE) since version 4 but wasn't standardized until years later with CSS3 web fonts. Before web fonts gained popularity and cross-browser support, web developers used to rely on all kinds of weird replacement techniques, such as image replacement and siFR (Scalable Inman Flash Replacement—see http://en.wikipedia.org/wiki/Scalable_Inman_Flash_Replacement) if they wanted custom fonts in headings.
- Bulletproof CSS. Back in the late 1990s and early 2000s a lot of pioneering techniques started to spring up for creating CSS UI features that wouldn't break if the text was resized. The text wouldn't spill messily out of its containers; instead, the design would expand along with it. These techniques were referred to as "Bulletproof CSS," and they worked well if done properly. But often they required a number of nested `<div>`s, each with a single background image hung off it. Bulletproof rounded corners on a container required four nested `<div>`s! Such designs were inflexible as well. If you wanted to then change the color of the background, you'd have to go back into your preferred graphics editor and update all the background images each time. This is exactly the kind of problem that properties like `border-radius` were created to fix.

- Multiple column layouts. It is very common to use CSS floats to create multiple column layouts; this everyone knows. But this is somewhat of a hack. Floats were never originally intended for this purpose. They were intended for simple magazine layout image floats.
- Dynamic UIs. Many "dynamic UI features," such as layouts that automatically adapt to different screen widths and smooth animations and transitions for user feedback, have been traditionally done using JavaScript. There was no way to achieve them using CSS alone until recently; hence, the rise of DHTML in the late 1990s (yuck!) and more recently, the overwhelming popularity of JavaScript libraries, such as jQuery and Dojo.

And the list goes on. CSS3 was created not to give users a completely new set of amazing features to play with and create "spangly web innovations" (a great design agency name if ever there was one), but more to provide users with standardized, more flexible ways of solving existing problems.

There are now more than 40 modules in CSS3 at various stages of completion and browser support. The modular system is beneficial in many ways. It makes CSS3 easier to write by the spec teams and implement by the browser vendors: It is always easier to tackle small chunks than a single giant monolith. It also makes it easier for web designers and developers to get their heads around, and in my opinion, it makes it easier to "sell" to clients who may have issues about using "unfinished" technologies in their sites (yes, CSS 2.1 was technically only finished in 2011, but hey).

MODERN **WEB DESIGN** PHILOSOPHY

FIGURE 1.1
It's highly impressive to be able to create web pages like this, just using open standards (see http://operasoftware.github.com/Emberwind and http://helloracer.com/webgl).

I am a great supporter of CSS and the rest of the open standards landscape. The last couple of years have been very exciting for open standards. You've seen browsers leap forward in terms of rendering speed, feature support, and so on. New web technologies like CSS3 and HTML5/WebGL really do allow you to create some amazing digital experiences (**Figure 1.1**).

FIGURE 1.2 "Best viewed in Google Chrome" sounds like a step back to the days of "Best viewed in IE4." Now, I'm not saying that all content should be accessible to all people: It is not always that simple. But you should make such allowances whenever possible.

But everyone needs to take a step back when considering such innovations and not lose sight of the original qualities and best practices that made the web great, such as accessibility, usability, and graceful degradation.

ACCESSIBILITY COMES FIRST

In terms of my perspective on web design, I am really a "web 1.0" kinda guy. Innovative technologies are exciting, and you can fully appreciate their importance in the evolution of the web. But what is more exciting is the universal nature of the web. It's the fact that you can take the same content, style it in a million different ways, and still have it remain accessible to all web users the world over regardless of how they use the web—be it on a mobile phone, using only keyboard controls, or via a screen reader.

It is something designers and developers shouldn't lose sight of, but often we do. Whenever an exciting new web technology comes to the forefront, too many sites tend to pop up that go wild with the shiny and forget about the basic tenets. Recently, you've seen a sad reemergence of "This site is best viewed in..." messages, which should have been eliminated after the original browser wars ended a decade or so ago. And what about important text content rendered in `<canvas>`, which is therefore inaccessible? And how about CSS3 features that could work across multiple browsers but don't because the designer has only used the `-webkit-` prefixed version of the property? That designer might say, "Oooh, but it's an app; therefore, it's important to lock out anyone who isn't using a device of the correct level of shininess" (**Figure 1.2**).

USABILITY NEXT!

Once your users have managed to access your content and services, can they make sense of it and glean the information they wanted from it? This is a simple, perhaps obvious point to make, but I've lost count of the times I've gone to a company website and scratched my head in vain while trying to find contact details, opening times, or an address. Instead, I find nothing useful amidst the sea of marketing BS, cheesy videos, and other propaganda being presented.

Why do people not think more about what information is most useful to people viewing their websites and how to present that information in an easily digestible way? A simple, well-written, and clearly available bit of copy is nearly always more effective than reams of flashy, whizzy, technical stuff.

My mantra for usability (and many other people's, too) is "don't make me think." Don't make your users think about how to get what they want. If you've not already read it, Steve Krug's book *Don't Make Me Think: A Common Sense Approach To Web Usability, 2nd Edition* is essential reading.

GRACEFUL DEGRADATION AND PROGRESSIVE ENHANCEMENT

Graceful degradation and progressive enhancement were two terms that first became popular (or at least noteworthy) about a decade ago. Both were used when talking about what happens to content when the browser viewing it doesn't support all the features used to create it.

Graceful degradation means that the content falls back to something simpler but still perfectly accessible and usable. So, for example, if a content box is built and then styled using lots of CSS3 glitz, older browsers should still be able to display the text in a readable form, even if it doesn't look as nice.

Progressive enhancement means that the base content is accessible by all, but then usability and stylistic enhancements are built on top of that base for those browsers that support those enhancements.

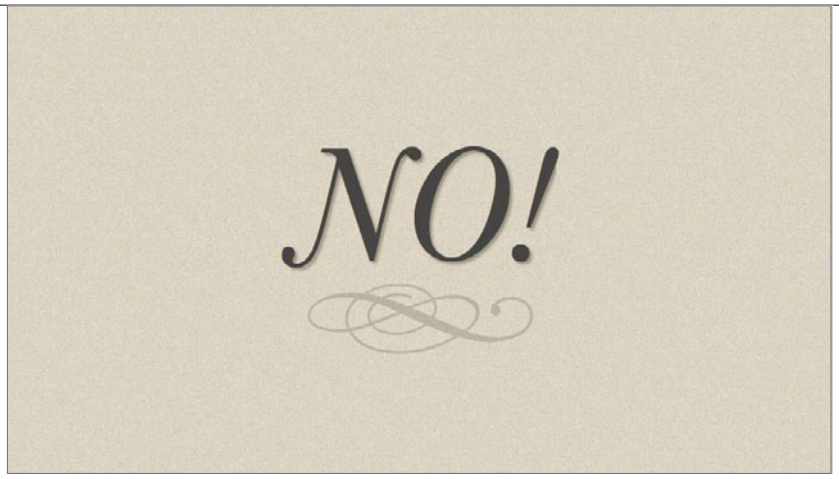


FIGURE 1.3 Dan Cederholm said it best with this cheeky little site.

These are design philosophies that I have always held dear. They have not always been easy to uphold, because you often meet clients who are "obsessed with pixel perfection across all browsers" or some similar weird fetish. But they are certainly becoming cool again, especially with all the CSS3 features to make use of and lots of mobiles and other alternative browsing devices to make your content work across. Oh, and IE6, 7, and 8 still have significant market share and often need to be supported.

The wide variety of new devices you have to support these days (mobile phones, tablets, TVs, etc.) actually makes things easier in terms of clients craving pixel perfection across all devices: It is impossible for sites to look and function the same across all desktop and mobile platforms, and indeed it doesn't make sense (as aped by dowebsitesneedtolookexactlythesameineverybrowser.com, seen in **Figure 1.3**). It is all about context. What makes sense on a standard desktop computer might well provide a bad user experience on a touchscreen mobile device or tablet.

The good news is that CSS3 is fairly easy to progressively enhance and gracefully degrade, and otherwise get to work OK across old browsers. Most of the features, if used in the right way, will degrade gracefully so that the base content will still be accessible in nonsupporting browsers. Also, there are mechanisms that allow you to build in support or provide alternative content if need be.

THOUGHT PROCESS FOR CONTENT

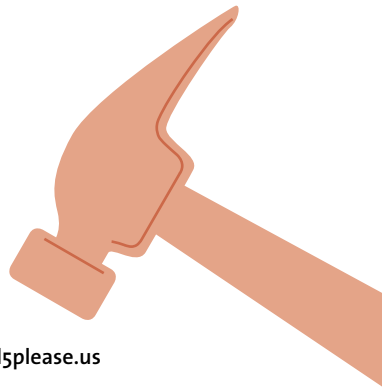
A good thought process to go through when implementing shiny features on a website interface is as follows:

1. Create a base of accessible HTML content. The styling and behavior you build on top of this content should, wherever possible, be usability and stylistic enhancements, and not essential for accessing the content.
2. Consider whether you need to use all the cool, cutting-edge technologies or whether you just want to because you're a cool kid who wants to be in with the in crowd.
3. Check whether your proposed implementation will gracefully degrade while leaving the base content accessible.
4. Test whether the content is accessible and OK looking across varying devices (e.g., different screen sizes, control mechanisms).
5. In cases where the content is not accessible without the CSS3, WebGL, or whatever, or not accessible to certain users, do your best to build in alternative mechanisms that will provide access to that content.

You should constantly look at making content work for as many users as possible by:

1. Keeping graceful degradation/progressive enhancement in mind.
2. Providing alternatives for inaccessible content using built-in features (e.g., alt text, transcripts for video).
3. Building in your own alternatives when no built-in mechanisms exist (e.g., feature detection and provision of alternative styles using Modernizr).
4. Using polyfills to provide support for features where none exists.

The rule I used for deciding what to cover in this book was to include a CSS3 feature only if it has support across at least two major browsers and if you can make designs employing it work in older browsers that don't support it via polyfills, alternative content, graceful degradation, and so on. I've broken this rule a few times, but only when I thought a feature was very significant and likely to have more implementations soon, and when nonsupport didn't completely break sites.



TIP: A great site to consult for quick summaries of which CSS3 and HTML5 features are ready to use on production sites, and whether fallbacks and the like should be provided is http://html5please.us by Divya Manian, Paul Irish, et al.

CSS3 MODULES IN THIS BOOK

Let's look at a brief roundup of the major CSS3 modules you'll be utilising and their main features. You can find more details on the latest status of each module at the W3C CSS Current Work page atwww.w3.org/Style/CSS/current-work.en.html. As you'll see, many of the modules are not yet finished, but this shouldn't stop you from using some of those features. Many such features are already supported in browsers, albeit with vendor prefixes (see the section "Vendor Prefixes" for more details).

The major CSS3 modules featured in this book include:

- CSS Color (www.w3.org/TR/css3-color). CSS Color defines the many ways to specify color in CSS3, including RGB (red, green, blue), HSL (hue, saturation, lightness), RGBA and HSLA (same as before but includes an alpha channel to specify transparency), and a separate opacity property to apply transparency to a whole selection of elements.
- CSS Fonts Level 3 (www.w3.org/TR/css3-fonts). As well as containing the definitions for downloadable web fonts (previously in a separate module known as, you guessed it, CSS web fonts), this module also contains definitions for other font-affecting properties, such as `font-feature-settings`. I won't talk about many of these beyond web fonts, because many do not have much browser support yet. You'll mostly meet these in Chapter 3.
- CSS Text Level 3 (www.w3.org/TR/css3-text). This goes hand in hand with CSS Fonts Level 3 to give you more power over your words! As well as housing familiar items from CSS2, such as `letter-spacing` and `text-transform`, CSS Text introduces new friends, such as hyphenation and text shadow.
- Selectors Level 3 (www.w3.org/TR/css3-selectors). Selectors Level 3 defines a much more powerful, robust set of mechanisms for selecting the elements you want to apply styles to than was available in CSS2. Pretty much all of these selectors have good support across modern browsers. These are discussed later in the "CSS3 Selectors" section of this chapter.
- Media Queries (www.w3.org/TR/css3-mediaqueries). The primary means by which you can now serve optimized different layouts of the same content to widely differing browsing devices—for example, wide screen and narrow screen. You'll mostly meet these in Chapter 8.

- Backgrounds and Borders Level 3 (www.w3.org/TR/css3-background). Backgrounds and Borders defines anything to do with background and borders, including rounded corners (`border-radius`), drop shadows (`box-shadow`), and fancy border effects (`border-image`).
- CSS Multi-column layout (www.w3.org/TR/css3-multicol). CSS Multi-column layout defines an easy way to break up content into multi-column layouts that reflow nicely rather than having to hack it with imprecise floats. You'll meet these in Chapter 7.
- CSS transforms (www.w3.org/TR/css3-2d-transforms and www.w3.org/TR/css3-3d-transforms). These two specifications define mechanisms for transforming the size, position, and shape of elements in two and three dimensions. I'll mainly talk about these in Chapter 5.
- CSS transitions (www.w3.org/TR/css3-transitions). CSS transitions give you a way to smoothly animate changes in state, such as a change in link color or an increase in banner size on hover. You'll mainly see these in Chapter 5.
- CSS animations (www.w3.org/TR/css3-animations). CSS animations allow you to implement Flash-style declarative animations using keyframes detailing different property values, which the browser then "tweens" between. These are also covered in Chapter 5.
- CSS Flexible box layout (www.w3.org/TR/css3-flexbox). Mainly intended for equally distributing the height or width of rows or columns, this module defines new values for the `display` property to allow more powerful layout techniques. This is supported to varying degrees across modern browsers, but it is definitely worth mentioning.
- CSS Image Values and Replaced Content Level 3(www.w3.org/TR/css3-images). This module contains some useful features for controlling background images and replaced content, some of which is starting to be supported across browsers. I'll cover linear and radial gradients among other features.

GENERAL CSS3 FEATURES

To whet your appetite, let's now look at some general CSS3 features. These features are grouped together because they are general features that you'll meet time and time again throughout different chapters: They are useful in many different circumstances.

VENDOR PREFIXES

Vendor prefixes are not exactly specific CSS3 features, but at the time of this writing (and for some time after), you'll meet them repeatedly when working with CSS3. The reason is that many of the modules you'll be using features from aren't finished.

The idea is that before a CSS feature is completely "finished" (e.g., the spec is not quite stable, and changes may be made before the final version), it can still be implemented inside browsers. At this stage, browser vendors add their own vendor prefix to the start of the feature and use the prefixed version. This allows each vendor to support the feature inside its own "sandbox," as it were, so if the spec changes and future versions work differently, this won't result in a single property that works differently across different browsers. As an example, CSS transitions are currently supported across browsers with vendor prefixes. A sample block of code might look like this:

```
a:link {
    background-color: #666666;
    -webkit-transition: 1s all;
    -moz-transition: 1s all;
    -ms-transition: 1s all;
    -o-transition: 1s all;
    transition: 1s all;
}
a: hover {
    background-color: #ffffff;
}
```

A PREFIXED **NIGHTMARE?**

I've put the prefixed properties in my example in the order they are in for two reasons. First, it looks aesthetically pleasing to have the widest prefix first and the narrowest last.

Second, at the time of this writing, a number of non-WebKit browser makers were discussing adding support for `-webkit-` prefixed versions of some properties, as well as their own prefixed versions. By putting -webkit- first, you can make sure that if this happens, such browsers will end up using their own prefixed version if it is present, not `-webkit-`, because the others all appear afterwards in the cascade.

Using the correct prefixed property will always be better and more accurate than relying on faked `-webkit-` support, especially considering that in some cases you might feed the different browsers different property values because of varying support. For example, at the time of this writing Opera does not yet support 3D transforms, so you could provide Opera with this 2D transform that would work:

```
-webkit-transform: rotate3D(1,0,0,10deg) translateX(300px);
-o-transform: translateX(300px);
```

Why are other browsers considering adding `-webkit-` support? Because so many developers have been harboring an ill-conceived idea that WebKit is the only browser engine worth supporting. So they were using lots of CSS3 features only with the `-webkit-` prefix, thereby making those features arbitrarily fail in other browsers that support them. As far as users are concerned, it is the browsers that are at fault. The average site visitors don't know any better, and neither should they be expected to. Messy as it is, non-WebKit browsers adding `-webkit-` support is a somewhat desperate potential measure to try to fix this browser support mess to some degree.

To sum up, it may sound nightmarish having to include five different versions of the same property in such situations. Quite a few people think that you shouldn't use vendor prefixes at all in production projects, and that they are only for testing purposes (this is the W3C's official stance too). But don't let that stop you. As you'll discover throughout this book, it is easy in most cases to retain an acceptable user experience in browsers that don't support those properties, as long as you give it a bit of forethought!

If you want to use prefixed CSS3 features, please do so responsibly and use all the different prefixes for all supporting browsers. And don't make your sites dependent on a particular feature that doesn't have cross-browser support!

This `transition` shorthand property tells the browser to smoothly transition every property that changes when the link's state changes over a duration of 1 second (see Chapter 5 for more details). In this case it is just the background color that changes. The aspect to focus on in this code is the fact that there are five copies of the `transition` property. The first four include vendor prefixes. At the time of