THE EXPERT'S VOICE® IN WEB DEVELOPMENT

CSS Mastery

Advanced Web Standards Solutions

Third Edition

Andy Budd
Emil Björklund

Apress®

CSS Mastery

Advanced Web Standards Solutions

Third Edition

Andy Budd

Emil Björklund

Apress®

CSS Mastery: Advanced Web Standards Solutions, Third Edition

Andy Budd
Brighton, United Kingdom

Emil Björklund
Malmo, Sweden

ISBN-13 (pbk): 978-1-4302-5863-6

ISBN-13 (electronic): 978-1-4302-5864-3

DOI 10.1007/978-1-4302-5864-3

Library of Congress Control Number: 2016944612

Managing Director: Welmoed Spahr
Acquisitions Editor: Ben Renow-Clarke
Development Editor: Matthew Moodie
Technical Reviewers: Anna Debenham and Andy Hume
Editorial Board: Steve Anglin, Pramila Balen, Louise Corrigan, James DeWolf, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Nancy Chen
Copy Editor: Bill McManus
Compositor: SPi Global
Indexer: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 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 www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

This book is dedicated to all my colleagues at Clearleft—both past and present. Were it not for their support and wisdom, this book would never have happened.

—Andy Budd

Dedicated to the memory of my grandfather: the engineer, artist, and life-long tinkerer Sven Forsberg (1919–2016).

—Emil Björklund

Contents at a Glance

Contents

About the Authors

Andy Budd is one of the founding partners at digital design consultancy, Clearleft. An early champion of web standards in the UK, the first edition of this book played a small but important role in the eventual adoption of CSS. These days, Andy is primarily focused on advancing the field of user experience design. He does this by consulting with clients, writing articles, mentoring new designers, and speaking at events like SXSW, An Event Apart, and The Next Web. Andy founded the long running dConstruct conference, and currently curates the popular UX London conference. In 2011 Andy co-founded the Brighton Digital Festival, which now sees 40,000 visitors attend 190 events in the city of Brighton each September.

As an active member of the design community, Andy has helped judge a number of international design awards, and is a mentor at Seedcamp. He is also the driving force behind Silverback, a low-cost usability-testing tool for the Mac. An avid Twitter user, Andy occasionally finds time to blog at `andybudd.com`.

Never happier than when he's diving in some remote tropical atoll, Andy is a qualified PADI dive instructor and retired shark wrangler.

Emil Björklund is the Technical Director at digital design consultancy inUse, where he's usually busy building websites—or helping clients and co-workers better their craft. Emil created his first HTML page on Geocities in 1997, but got confused by the mess of table tags. Coming back to the Web in 2001, he found this magical thing called CSS and has been fascinated by it ever since.

For the last decade, Emil has been building websites professionally, hacking on anything from client-side JavaScript to server-side Python, but always with a special place in his heart for good old HTML and CSS. Emil's writing and advice on CSS has been published in Net Magazine and on CSS Tricks. He also writes about (mostly) web-related stuff on his blog at `thatemil.com`.

Emil lives with his girlfriend and their cat in Malmö, Sweden, far from any sharks.

About the Technical Reviewers

Andrew Hume is a web developer from Brighton in the UK. He's spent the last fifteen years leading front-end teams for websites like Twitter, Bing, and The Guardian. As a consultant for Clearleft, he helped figure out large, scalable CSS systems for clients like the BBC, eBay, and Mozilla.

Anna Debenham is a Freelance Front-end Developer living and working in London in the UK. In 2013, she was awarded Netmag's Young Developer of the Year award. She's the author of Front-end Style Guides, a Technical Editor for A List Apart, and every December, she co-produces 24 Ways.

Acknowledgments

We would like to thank the tireless work of Jeffrey Zeldman, Eric Meyer, and Tantek Çelik, without whom the web standards movement would never have happened. We'd like to thank those who followed. People like John Allsopp, Rachel Andrew, Mark Boulton, Doug Bowman, Dan Cederholm, Andy Clarke, Simon Collison, Jon Hicks, Molly E. Holzschlag, Aaron Gustafson, Shaun Inman, Jeremy Keith, Peter-Paul Koch, Ethan Marcotte, Drew McLellan, Cameron Moll, Dave Shea, Nicole Sullivan, and Jason Santa-Maria, who answered the challenge and helped take CSS mainstream. Finally, we'd like to thank all those tireless designers and developers who have subsequently picked up the baton, and have helped turn CSS into the modern design language we know today. There are too many to list everybody, but some of the people who have made the biggest impact on our practice in later years include Chris Coyier, Vasilis van Gemert, Stephen Hay, Val Head, Paul Lewis, Rachel Nabors, Harry Roberts, Lea Verou, Ryan Seddon, Jen Simmons, Sara Soueidan, Trent Walton, and Estelle Weyl. We'd also like to thank all the designers and developers who constantly help and inspire us by bouncing ideas about CSS on Twitter and various Slack teams.

We want to thank everybody who helped get this book over the finish line, including inUse who sponsored part of the work. A special thanks to technical editor Anna Debenham—if there are any errors in the book, it's more than likely we put them in there when she was looking the other way. We'd also like to thank Andy Hume, who contributed his expertise during the early phases of writing, setting the direction for this new edition. Furthermore, we'd like to thank Charlotte Jackson, Peter-Paul Koch, Paul Lloyd, Mark Perkins, and Richard Rutter for reading early drafts, bouncing ideas, and giving invaluable feedback.

Photos in the book or in examples are in most instances taken by us or gathered from Public Domain sources. The following images are licensed via the Creative Commons Attribution 2.0 license (https://creativecommons.org/licenses/by/2.0/): "Portrait" by Jeremy Keith (https://flic.kr/p/dwFRgH) and "A Long Night Falls on Saturn's Rings" by NASA Goddard Space Flight Center (https://flic.kr/p/7ayNkz).

Finally, we'd both like to thank our partners for patience and support during the considerable time it took to produce these pages.

Introduction

When I started writing the first edition of *CSS Mastery* way back in 2004, there were already two CSS books in the market, so I wasn't sure the world needed a third. After all, CSS was still a relatively niche subject back then; largely the preserve of bloggers and web standards enthusiasts. The majority of sites were still being built using tables and frames, and the folks on my local developer mailing list thought I was mad, and CSS was just a pipe dream. Little did they know we were on the verge of a web standards revolution and the field exploded around the time the book was published, pushing the book to the top of my publisher's bestselling chart for years to come.

By the time the second edition came out, CSS was now firmly established. The role of the book changed from exposing new people to the power of CSS, to helping make them more efficient and effective. So we scoured the Web for the latest techniques, workarounds, and hacks, and created a book we hoped would become the definitive guide for web designers and front-end developers everywhere. It felt like we'd reached a stable point in the development of the language, and the book would remain relevant for a long time. How wrong we were.

Rather than becoming stagnant, CSS of recent years feels like it has finally started to live up to its original promise. We entered the golden age of web standards; an age where browser support was good enough for us to finally move focus away from hacks, instead putting our efforts into writing elegant, well-crafted, and highly maintainable code for the largest and most complicated sites around.

So it was time to write a third edition; to bring together all these new tools, techniques, and ways of thinking into a single reference. To help in this task I drew upon the skills of my good friend, Emil Björklund, a developer of rare skill and ability. What Emil brings to the book is a deep understanding of modern CSS practices; how to craft highly flexible code using the latest techniques that works across the widest range of browsers, screens, and platforms, in the most elegant way possible.

Together we've almost completely rewritten the book from the ground up, adding new chapters on web typography, animation, layout, responsive design, how to structure your code, and much more. This new edition follows in the footsteps of previous editions, offering a mix of practical examples, language reference, and cross-browser workarounds for tricky techniques. The sign of CSS mastery is no longer about knowing all the arcane hacks to make CSS work at all, or knowing all properties by heart. CSS today consists of several dozen specifications, encompassing hundreds of properties—there's probably no one who knows it all! Instead, this book emphasizes flexibility and robustness, making sure your code works in the ever-changing landscape of different browsers, devices, and usage situations. We won't cover every single language feature, but you will find a good overview of what's available, some lesser-known old-school tricks, and the occasional glimpse into the future of CSS.

To enjoy the book fully, you should have at least some small grasp of how CSS works—maybe you have played with it for a while, or even worked on a website or two. The book starts with three short introductory chapters on the very foundations of creating and styling web pages, so even if you're rusty, you'll get a recap. After that, each chapter introduces new features of the language and progressively more complex examples. Even if you're a seasoned CSS practitioner, you should find plenty of interesting and useful techniques for solving common web design problems, in which case you should feel free to jump to the chapters that pique your interest.

Regardless of your previous understanding of the language, we hope the resulting book will help you unlock some of the secrets of CSS and become a true CSS Master.

CHAPTER 1

Setting the Foundations

The human race is a naturally inquisitive species. We just love tinkering with things. When we got our new Parrot AR Drone at the office, we had it in pieces before we'd even looked at the instructions. We enjoy working things out ourselves and creating our own mental models of how we think things behave. We muddle through and only turn to the manual when something goes wrong or defies our expectations.

One of the best ways to learn Cascading Style Sheets (CSS) is to jump right in and start tinkering. In fact, this is likely how many of you learned to code; by picking up tips from blogs, viewing source to see how your favorite designers had achieved a particular effect, and by browsing open source repositories for snippets of code. You almost certainly didn't start out by reading the full specification, which is enough to put anyone to sleep.

Tinkering is a great way to start, but if you're not careful, you may end up misunderstanding a crucial concept or building in problems for later on. We know; we've done so several times. In this chapter, we're going to review some basic but often misunderstood concepts and show you how to keep your HTML and CSS clear and well structured.

In this chapter you will learn about:

- The importance of maintainability
- Different versions of HTML and CSS
- Strategies for future-friendly and backward-compatible code
- Adding meaning to your HTML and using newer HTML5 elements
- Adding appropriate styling hooks to HTML
- Extending HTML semantics with ARIA, microformats, and microdata
- Browser engine modes and validation

Structuring your Code

Most people don't think about the foundations of a building. However, without solid foundations the majority of buildings wouldn't stay standing. While this book is about CSS techniques and concepts, much of what you are about to learn would not be possible (or at least would be very difficult) without a well-structured and valid HTML document to work with.

In this section you will learn why well-structured and meaningful HTML is vital to standards-based development. You will learn how you can add more meaning and flexibility to your documents, and by doing so, make your job as a developer easier. But first up is a topic of the utmost importance no matter what language we happen to be working in.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4302-5864-3_1) contains supplementary material, which is available to authorized users.

A. Budd and E. Björklund, *CSS Mastery,* DOI 10.1007/978-1-4302-5864-3_1

Maintainability

Maintainability is arguably the most important characteristic of any good code base. If your code begins to lose structure and becomes hard to read, then lots of things become difficult. Adding new features, fixing bugs, and improving performance all become more complicated and frustrating if you're struggling with unreadable and brittle code. In some cases it gets so bad that developers will resist making changes altogether, because nearly every time they do, something breaks. This can lead to a situation where no one enjoys working on the website or, in very bad circumstances, to a strict change control process where releases can only be carried out once a week or even once a month!

If you are building websites that are to be handed off to a client or another development team, maintainability is even more important. It's critical that you provide code that is easy to read, explicit in its intent, and *optimized for change*. "The only constant is change" is a particularly appropriate cliché to invoke here, because whose project doesn't have continually changing requirements, along with constant feature requests and bug fixes?

CSS is one of the hardest languages to keep maintainable as a codebase grows, and the style sheets for even a relatively small site can get out of hand quickly. Other modern programming languages have features like variables, functions, and namespaces built in; all features which help keep code structured and modular by default. CSS doesn't have these features, so we need to build them into the way that we use the language and structure our code. As we discuss different topics throughout the book, you'll see the theme of maintainability evident across nearly all of them.

A Brief History of Markup

> *The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.*
>
> —Tim Berners-Lee

Tim Berners-Lee created HTML in 1990, for the purpose of formatting scientific research documents. It was a simple markup language that enabled text to be given basic structure and meaning, such as headings, lists, and definitions. These documents were typically presented with little or no visual embellishment, and could be easily indexed by computers and read by people using a text-only terminal, a web browser, or screen reader if necessary.

However, humans are very visual creatures, and as the World Wide Web gained in popularity, HTML started to acquire features for creating presentational effects. Instead of using heading elements for page headlines, people would use a combination of font and bold tags to create a specific visual effect. Tables got co-opted as a layout tool rather than a way of displaying data, and people would use `blockquote` elements to indent text rather than to indicate quotations. Very quickly HTML lost its primary purpose of giving structure and meaning to content, and became a jumble of `font` and `table` tags. Web designers came up with a name for this kind of markup; they called it *tag soup* (see Figure 1-1).

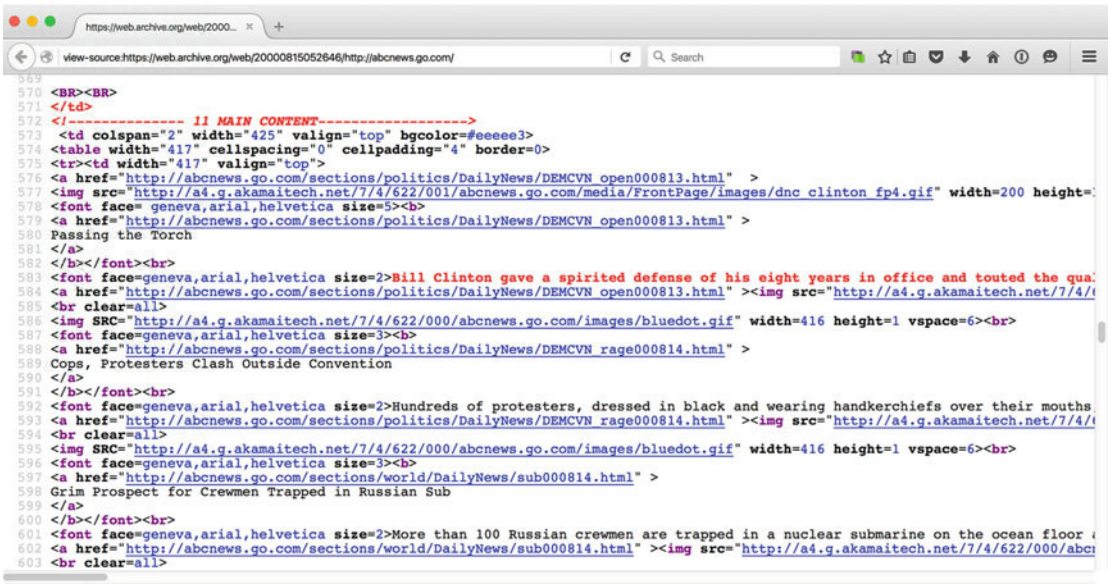

Figure 1-1. *The markup for the lead story from abcnews.com on August 14, 2000, uses tables for layout and large, bold text for headings. The code lacks structure and is difficult to understand*

The Web had become a mess, and CSS was created to help tidy things up. The primary purpose of CSS was to allow the presentational rules that had found their way into HTML to be extracted and put into their own system; to *separate content and presentation*. This encouraged meaning and semantics to creep back into HTML documents. Presentational tags like the `font` tag could be ditched, and layout tables could be slowly replaced. This was a boon for the accessibility and speed of much of the Web, but CSS also provides a number of benefits for web designers and developers:

- A language specifically designed to control visual style and layout
- Styles that can be more easily reused across one site
- Improved code structure through separation of concerns

SEPARATION OF CONCERNS

The concept of separation of concerns is common in software development. On the Web, it can be applied not only to the separation of markup and style, but also to how the styles are written. In fact, it's one of the main methods for ensuring maintainable code.

There's a phrase common in the Unix development community that expresses this concept through the mantra "Small pieces, loosely joined." A "small piece" is a focused module of code that does one thing really well. And because it is "loosely joined" to other components, that module can be easily reused in other parts of the system. A "small piece" in Unix could be a word count function, which will work with any piece of text you feed into it. In web development, a "small piece" could be a `product-list` component, which if loosely coupled will be reusable across multiple pages of a site, or in different parts of the layout.

You could think of these small pieces of code as LEGO bricks. Each brick is incredibly simple, but they can be joined together in numerous ways to create objects of immense complexity. Towards the end of the book, in Chapter 12, we will return to this topic, and examine how to use this strategy in a structured way.

Different Versions of HTML and CSS

CSS comes in various versions, or "levels", and it's good to have some historical context around what these versions mean and how they impact what features of CSS you should or shouldn't use. The World Wide Web Consortium (W3C) is the organization that looks after standardizing web technology, and each of its specifications goes through a number of phases of development before it finally becomes a W3C recommendation. CSS 1 became a W3C recommendation at the end of 1996 and contains very basic properties such as fonts, colors, and margins. CSS 2 became a recommendation in 1998 and added advanced concepts such as floating and positioning, as well as new selectors like the child, adjacent sibling, and universal selectors.

CSS 3 is a slightly different beast. In fact, there *is* no CSS 3 as such, but a collection of modules each *leveled* independently. When a module specification continues to improve an existing concept, it starts at level 3. If it's an entirely new technology it starts at level 1. When the term CSS 3 is used, it usually refers to anything new enough to be part of a module specification. Examples of modules include "CSS Backgrounds and Borders Level 3," "Selectors Level 4," and "CSS Grid Layout Level 1." This modular approach allows different specifications to move at different speeds, and some level 3 specifications, such as "CSS Color Level 3", have already been published as recommendations. Others are in candidate recommendation status, and many are still in working draft status.

Although work began on CSS 3 at about the time CSS 2 was published, progress on these new specifications was initially slow. For that reason, in 2002, the W3C published CSS 2 Revision 1. CSS 2.1 fixes errors in CSS 2, removes features that were poorly supported or nonexistent in browsers, and generally cleans things up to provide a more accurate picture of browser implementations. CSS 2.1 reached recommendation status in June 2011, over a decade after work on CSS 3 had started. That gives you an idea of how long it can take for standards bodies and browser makers to nail down exactly how these technologies work. That being said, browsers often ship experimental support for features when the features are in the draft stage, and at the candidate recommendation stage things are usually quite stable. The date when something becomes a usable technology is usually much earlier than the date when it becomes a recommendation.

The history of HTML is no less complex. HTML 4.01 became a recommendation in 1999, at which point the W3C switched its attention to XHTML 1.0. XHTML 1.1 was meant to follow, but the level of strictness it imposed proved impractical and it fell out of favor among members of the web development community. Essentially, progress on the main language of the Web stalled.

In 2004, a group of companies formed the Web Hypertext Application Technology Working Group (WHATWG) and started working on a new set of specifications. The W3C acknowledged the need for this work in 2006, and joined in on the fun. In 2009, the W3C gave up on XHTML completely, and formally embraced the new standard from the WHATWG that had become known as HTML5. Initially, both the WHATWG and the W3C harmonized their work on the standards, but somewhere along the line, their relationship status turned complicated. Today, they edit two separate standards, the one from the WHATWG being known as just HTML, and the one from the W3C known as HTML5. Yes, we know, it's all a bit nuts. Fortunately, the two standards are very close to each other, so speaking of HTML5 as a single thing still makes sense.

What Version Should I Use?

Designers and developers often ask which version of HTML or CSS they should use, but there's no simple answer to this. Although the specifications provide a focal point for standards and the work that goes on in developing web technology, they are largely irrelevant to designers and developers on a day-to-day basis. What *is* important is knowing what parts of HTML and CSS have been implemented in browsers, and how robust and bug-free those implementations are. Are they experimental features that should be used with caution? Or are they robust and well tested, with matching implementations across a large number of browsers?

Understanding the state of browser support is one of the trickiest parts of writing CSS and HTML today. Sometimes it can seem like things move very fast, and you have to work hard to keep up. At other times it can feel frustratingly slow. Throughout this book you will see browser support notes for various features of HTML and CSS, along with tips as to how and when you should consider using them. Inevitably the information printed here will become outdated, so it is important to keep up with this information yourself.

There are several good places to start learning about browser support. For CSS properties, the "Can I use" website (`http://caniuse.com`) allows you to search for a property or suite of properties, complete with statistics on what percent of browsers support it, across both desktop and mobile browsers. Another ambitious initiative is `http://webplatform.org`, a collaboration between the W3C and several browser makers and industry giants, attempting to collect and merge all their respective documents on support for CSS, HTML, JavaScript APIs, and so forth. However, as large projects are prone to do, putting together this canonical web technology documentation is taking a lot of time. While that's happening, Mozilla's developer documentation, MDN (`http://developer.mozilla.org`), is generally considered the gold standard.

When discussing browser support, it's important to accept that not all browsers are created equal; and they never will be. Some CSS 3 features are supported by very few browsers today. For example, Flexible Box Layout (or *flexbox* for short) was not correctly supported in Internet Explorer until version 11 and in Safari until version 6.1. Even if you need to support legacy browsers, it does not mean that flexbox is of no use at all. You might avoid using flexbox for the core layout of your site, but you could still choose to use it in a specific component where its powerful features are extremely useful, and make sure that there is an acceptable fallback in browsers that don't understand the properties. The ability to make judgment calls on backward compatibility vs. future-friendly code is part of what defines the true CSS Master.

Progressive Enhancement

The ability to balance backward compatibility with the latest HTML and CSS features involves a strategy known as *progressive enhancement*. What this stands for is basically "start by making it work well for the lowest common denominator, but feel free to take things further where they are supported." Using progressive enhancement means you'll write your code in "layers," where each successive enhancement is only applied if it's supported or deemed appropriate. This may sound complicated, but the good news is that both HTML and CSS partly have this built in.

For HTML, this means that unknown elements or attributes generally cause no trouble for the browser; it will gobble them up without complaining but might not apply the resulting changes to how the page works. As an example, you could use the new types of `input` elements that are defined in HTML5. Say you have a form field for an e-mail address that's marked up like this:

```
<input type="text" id="field-email" name="field-email">
```

You could change the value of the `type` attribute like this:

```
<input type="email" id="field-email" name="field-email">
```

Browsers that haven't implemented the new field types will simply respond "I have no idea what that means" and fall back to the default type value, which is `"text"`, just like in the first example. Newer browsers that do understand the `"email"` type will know what kind of data the user is supposed to enter into this field. On many mobile devices, the software keyboard will adjust to show a view optimized for inputting e-mail addresses, and if you're using the built-in support for form validation in newer browsers, this will pick up on that too. We have *progressively enhanced* the page, with no downside for users on older browsers.

Another simple change is to update the document type declaration to the new, shorter version from the HTML5 standard. The document type, or doctype for short, is the bit at the top of an HTML document that's supposed to be a machine-readable hint about the version of the markup language used in the document. It used to be a long and complicated affair in older versions of HTML and XHTML, but in HTML5 it's been simplified down to just this:

```
<!DOCTYPE html>
```

You can safely switch to writing your HTML documents with this doctype because the HTML5 syntax and doctype are backward compatible. We'll have a closer look at some of the new elements available in HTML5 in upcoming sections, but if you need more in-depth information on how to start writing HTML5 markup today, check out Jeremy Keith's *HTML5 for Web Designers* at http://html5forwebdesigners.com.

Progressive enhancement in CSS works in a similar manner when it comes to how the browser interprets new properties. Any property or value that the browser doesn't recognize causes it to discard that declaration, so adding new properties has no ill effects as long as you provide a sensible fallback.

As an example, many modern browsers support the `rgba` functional notation for color values. It allows you to specify colors using separate values for the red, green, and blue channels as well as a transparency value, called the alpha channel. We can use it like this:

```
.overlay {
    background-color: #000;
    background-color: rgba(0, 0, 0, 0.8);
}
```

This rule states that elements with the `overlay` class name should have a black background color, but then immediately redeclares the background color to be a slightly transparent black using `rgba`. For browsers that don't understand the `rgba` notation, the second statement will be ignored, and the element will have a solid black background color. For browsers that *do* understand the `rgba` notation, the second statement overwrites the first. So even if `rgba` notation isn't supported everywhere, we can still use it, provided we use a fallback declaration that comes first.

Vendor Prefixes

Browser makers use the same principle to introduce experimental features into their browsers. They do this by prefixing the property name or value with a special string, so that only their own browser engine will apply it and other browsers will ignore it. This allows browser makers to introduce new features while the specifications are missing or immature. Style sheet authors can try them out without risk of breaking their pages if the different browsers interpret new features differently. For example:

```
.myThing {
  -webkit-transform: translate(0, 10px);
  -moz-transform: translate(0, 10px);
  -ms-transform: translate(0, 10px);
  transform: translate(0, 10px);
}
```

This applies a transformation to the element (something we will look at in Chapter 10) with a couple of different prefixes. Those starting with `-webkit-` apply to the WebKit-based browsers such as Safari. Google Chrome and Opera are based on the Blink engine, which in turn was initially based on WebKit, so the `-webkit`-prefix often works for them as well. The `-moz-` prefix applies to Mozilla-based browsers like Firefox, and the `-ms-` prefix applies to Microsoft's Internet Explorer.