

The
Pragmatic
Programmers

HTML5 & CSS3

Develop with Tomorrow's
Standards Today



Brian P. Hogan

Edited by Susannah Davidson Pfalzer



Under Construction

The book you're reading is still under development. As part of our Beta book program, we're releasing this copy well before a normal book would be released. That way you're able to get this content a couple of months before it's available in finished form, and we'll get feedback to make the book even better. The idea is that everyone wins!

Be warned. The book has not had a full technical edit, so it will contain errors. It has not been copyedited, so it will be full of typos and other weirdness. And there's been no effort spent doing layout, so you'll find bad page breaks, over-long lines with little black rectangles, incorrect hyphenations, and all the other ugly things that you wouldn't expect to see in a finished book. We can't be held liable if you use this book to try to create a spiffy application and you somehow end up with a strangely shaped farm implement instead. Despite all this, we think you'll enjoy it!

Download Updates

Throughout this process you'll be able to download updated ebooks from your account on <http://pragprog.com>. When the book is finally ready, you'll get the final version (and subsequent updates) from the same address.

Send us your feedback

In the meantime, we'd appreciate you sending us your feedback on this book at <http://pragprog.com/titles/bhh5/errata>, or by using the links at the bottom of each page.

Thank you for being part of the Pragmatic community!

► **Your Publishers, Andy & Dave**

HTML5 and CSS3

Develop with Tomorrow's Standards Today

Brian P. Hogan

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://www.pragprog.com>.

Copyright © 2010 Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-10: 1-934356-68-9

ISBN-13: 978-1-934356-68-5

Printed on acid-free paper.

B2.0 printing, July 19, 2010

Version: 2010-7-19

Contents

Change History	7
Beta 2—July 19th 2010	7
Preface	8
HTML5: The Platform vs The Specification	8
How This Works	9
What's In This Book	10
Prerequisites	10
Online Resources	11
1 An Overview of HTML5 and CSS3	12
1.1 A Platform for Web Development	12
1.2 Backwards Compatibility	15
1.3 The Road To The Future is Bumpy	15
Part I—Improving User Interfaces	19
2 New Structural Tags and Attributes	20
Tip 1 Redefining a Blog using Semantic Markup	23
Tip 2 Showing Progress with the Meter Element	34
Tip 3 Creating Popup Windows with Custom Data Attributes	38
3 Creating User-friendly Web Forms	43
Tip 4 Describing Data with New Input Fields	46
Tip 5 Jumping to the First Field with Autofocus	53
Tip 6 Providing Hints with Placeholder Text	55
Tip 7 In-Place Editing with ContentEditable	61
4 Making Better User Interfaces with CSS3	68
Tip 8 Styling Tables With Pseudo Classes	70
Tip 9 Making Links Printable with :after and content	79
Tip 10 Creating Multi-Column Layouts	83

5	Improving Accessibility	90
	Tip 11 Providing Navigation Hints with ARIA Roles	92
	Tip 12 Creating An Accessible Updatable Region	97
 Part II—New Sights And Sounds		103
6	Drawing On The Canvas	104
	Tip 13 Drawing A Logo	105
	Tip 14 Graphing Statistics with RGraph	112
7	Embedding Audio and Video	120
	Tip 15 Playing Sound Samples with the Audio tag	121
	Tip 16 Building a Cross-Platform Video Tutorial Page	122
8	Eye Candy	123
	Tip 17 Rounding Rough Edges	125
	Tip 18 Working With Shadows, Gradients, and Transformations	132
	Tip 19 Using Real Fonts	142
 Part III—Beyond HTML5		148
9	Working with Client-side Data	149
	Tip 20 Saving Preferences with LocalStorage	152
	Tip 21 Storing Data in Client-Side Relational Database	158
10	Playing Nicely With Others	170
	Tip 22 Cross document Messaging	171
	Tip 23 Getting Chatty with Websockets	172
	Tip 24 Finding Yourself With Geolocation	173
11	Where To Go Next	174
12	jQuery Primer	175
A	Bibliography	176
	Index	177

Change History

The book you're reading is in beta. This means that we update it frequently. This chapter lists the major changes that have been made at each beta release of the book, with the most recent change first.

Beta 2—July 19th 2010

This release includes three new tips, which you'll find in Chapter 8, *Eye Candy*, on page 123. You'll learn how to add rounded corners to form fields, work with shadows, transformations, and gradients, and you'll see how to work with fonts. As always, you'll see how you can accomplish these effects in browsers that don't offer native support.

You'll also find a small change to *Describing Data with New Input Fields*, on page 46. The newest versions of Safari and Chrome have broken our fallback detection for our colorpicker control, so we'll discuss how to work around that issue and how we can deal with situations like that in the future.

You'll find lots of small changes throughout the book as well, based on your valuable feedback in both the forums and in the book's errata. Please keep that coming!

Preface

Three months on the Web is like a year in real time.

Web developers pretty much think this way, since we're always hearing about something new. A year ago HTML5 and CSS3 seemed so far off in the distance, but already companies are using these technologies in their work today, as browsers like Google Chrome, Safari, Firefox, and Opera are starting to implement pieces of the specification.

HTML5 and CSS3 help lay the groundwork for the next generation of web applications. They let us build sites that are simpler to develop, easier to maintain, and more user-friendly. HTML5 has new elements for defining site structure and embedding content, which means we don't have to resort to extra markup or plugins. CSS3 provides advanced selectors, graphical enhancements, and better font support that makes our sites more visually appealing without using font image replacement techniques, complex JavaScript, or graphics tools. Improved accessibility support will improve AJAX applications for people with disabilities, and offline support lets us start building working application that don't need an Internet connection.

In this book, you're going to find out about all of the ways you can use HTML5 and CSS3 right now, even if your users don't have browsers that can support all of these features yet. Before we get started, let's take a second and talk about HTML5 and buzzwords.

HTML5: The Platform vs The Specification

HTML5 is a specification that describes some new tags and markup, and some wonderful JavaScript APIS, but it's getting caught up in a whirlwind of hype and promises. Unfortunately, HTML5 the standard, has evolved into HTML5, the platform, creating an awful lot of confusion among developers, customers, and even authors. In some cases, pieces

from the CSS3 specification like shadows, gradients, and transformations are being called “HTML”. Browser makers are trying to one-up each other with how much “HTML5” they support. People are starting to make strange requests like “My site will be in HTML5, right?”

For the majority of the book, we’ll focus on the HTML5 and CSS3 specifications themselves and how you can use the techniques they describe. In the last part of the book, we’ll look into a suite of closely related specifications that were once part of HTML5, which are in use right now on multiple platforms. These include Web SQL Databases, Geolocation, and WebSockets. While these things aren’t *technically* HTML5, they can help you build incredible things when combined with HTML5 and CSS3.

How This Works

Each chapter in this book focuses on a specific group of problems that we can solve with HTML5 and CSS3. Each chapter has an overview and a table summarizing the tags, features, or concepts covered in the chapter. The main content of each chapter is broken apart into “tips”, which introduce you to a specific concept, and walk you through building a simple example using the concept. The chapters in this book are grouped topically. Rather than group things into a HTML5 part and a CSS3 part, it made more sense to group them based on the problems they solve.

Each tip contains a section called “Falling Back”, which shows you methods for addressing your users who use browsers that don’t offer HTML5 and CSS3 support. We’ll be using a variety of techniques to make these fallbacks work, from third-party libraries to our own jQuery plugins. These tips can be read in any order you like.

Finally, each chapter wraps up with a section called “The Future” where we discuss how the concept can be applied as it becomes more widely adopted.

This book focuses on what you can use today. There are more HTML5 and CSS3 features that aren’t in use yet. You’ll learn more about those in the final chapter, Chapter 11, *Where To Go Next*, on page 174.

What's In This Book

We'll start off with a brief overview of HTML5 and CSS3, and take a look at some of the new structural tags you can use to describe your page content. Then we'll work with forms and you'll get a chance to use some of the form fields and features like autofocus and placeholders. From there, you'll get to play with CSS3's new selectors so you can learn how to apply styles to elements without adding extra markup to your content.

Then we'll explore HTML's audio and video support, and you'll learn how to use the canvas to draw shapes. You'll also get to see how to use CSS3's shadows, gradients, and transformations, and learn how to work with fonts.

Finally, we'll use HTML5's client-side features like LocalStorage, Web SQL databases, and offline support to build a simple application, we'll use WebSockets to talk to a simple chat service, and you'll see how HTML5 makes it possible to send messages and data across domains.

Prerequisites

This book is aimed primarily at web developers who have a good understanding of HTML and CSS. If you're just starting out, you'll still find this book valuable, but I recommend you check out *Designing With Web Standards* [], and my book, *Web Design For Developers*.

I also assume you have a basic understanding of JavaScript and jQuery¹, which we will be using to implement many of our fallback solutions. Chapter 12, *jQuery Primer*, on page 175 is a brief introduction to jQuery that covers the basic methods we'll be using.

You'll need Firefox 3.6, Google Chrome, Opera 10.6, or Safari 4 to test the code in this book. You'll probably need all of these browsers to test everything we'll be building, since each browser does things a little differently.

You'll also need a way to test your sites with Internet Explorer so you can ensure that the fallback solutions we create actually work. If you need to be able to test your examples in multiple versions of Internet Explorer, you can download IETester for Windows, as it supports IE 6,

1. <http://www.jquery.com>

7, and 8 in a single application. If you're not running Windows, you should consider using a virtual machine like VirtualBox or VMWare, or use a service like CrossBrowserTesting.

Online Resources

The book's website² has links to an interactive discussion forum as well as errata for the book. You can also find the source code for all of the examples in this book linked on that page. Additionally, readers of the eBook can click on the gray box above the code excerpts to download that snippet directly

If you find a mistake, please create an entry on the Errata page so we can get it addressed. If you have an electronic copy of this book, there are links in the footer of each page that you can use to easily submit errata.

Ready to go? Great! Let's get started with HTML5 and CSS3.

2. <http://www.pragprog.com/titles/bhh5/>

An Overview of HTML5 and CSS3

HTML5¹ and CSS3² are more than just two new standards proposed by the World Wide Web Consortium (W3C) and its working groups. They are the next iteration of technologies you use every day, and they're here to help you build better modern web applications. Before we dive into the deep details of HTML5 and CSS3, let's talk about some of the benefits of HTML5 and CSS3, as well as some of the challenges we'll face.

1.1 A Platform for Web Development

A lot of the new features of HTML center around creating a better platform for web-based applications. From more descriptive tags and better cross-site and cross-window communication to animations and improved multimedia support, developers using HTML5 have a lot of new tools to build better user experiences.

More Descriptive Markup

Each version of HTML introduces some new markup, but never before have there been so many new additions that directly relate to describing content. You'll learn about elements for defining headings, footers, navigation sections, sidebars, and articles in Chapter 2, *New Structural Tags and Attributes*, on page 20. You'll also learn about meters, progress bars, and how custom data attributes can help you mark up data.

1. HTML5 Specification: <http://www.w3.org/TR/html5/>

2. CSS3 is split across multiple modules, and you can follow its progress at <http://www.w3.org/Style/CSS/current-work>.

Multimedia with Less Reliance on Plugins

You don't need Flash for video, audio, and vector graphics anymore. Flash-based video players are relatively simple to use and work everywhere except for Apple's mobile devices, which has become a significant market. In Chapter 7, *Embedding Audio and Video*, on page 120 you'll see how to use HTML5 audio and video with effective fallbacks.

Better Applications

Developers have tried all kinds of things to make richer, more interactive applications on the web, from ActiveX controls to Flash. HTML5 offers amazing features that, in some cases, completely eliminate the need for third party technologies.

Cross-Document Messaging

Web browsers prevent us from using scripts on one domain to affect or interact with scripts on another domain. This restriction keeps end-users safe from cross-site scripting which has been used to do all sorts of nasty things to unsuspecting site visitors.

However, this prevents *all* scripts from working, even when we wrote them ourselves and we know we can trust the content. HTML5 includes a workaround which is both safe and simple to implement. You'll see how to make this work in *Cross document Messaging*, on page 171.

Web Sockets

HTML5 offers support for Web Sockets, which give you a persistent connection to a server. Instead of constantly polling a backend for progress updates, your web page can subscribe to a socket and the backend can push notifications to your users. We'll play with that a bit in *Getting Chatty with Websockets*, on page 172.

Client-side storage

We tend to think of HTML5 as a web technology, but with the addition of Local Storage and Web Databases, we can build applications in the browser that can persist data entirely on the client's machine. You'll see how to use those in Chapter 9, *Working with Client-side Data*, on page 149.

Better Interfaces

The user interface is such an important part of web applications, and we jump through hoops every day to make browsers do what we want.

In order to style a table or round corners, we either use JavaScript libraries or add tons of additional markup so we can apply styles. HTML5 and CSS3 make that practice a thing of the past.

Better Forms

HTML5 promises better user interface controls. For ages, we've been forced to use JavaScript and CSS to construct sliders, calendar date pickers, and color pickers. These are all defined as real elements in HTML5, just like dropdowns, checkboxes, and radio buttons. You'll learn about how to use these in Chapter 3, *Creating User-friendly Web Forms*, on page 43. While this isn't quite ready yet for every user, it's something you need to keep your eye on especially if you develop web-based applications. In addition to improved usability without reliance on JavaScript libraries, there's another benefit - improved accessibility. Screen readers and other browsers can implement these controls in specific ways so that they work easily for the disabled.

Improved Accessibility

Using the new elements in HTML5 to clearly describe our content makes it easier for software programs like screen readers to easily consume the content. A site's navigation, for example, is much easier to find if you can look for the `nav` tag instead of a specific `div` or unordered list. Footers, sidebars, and other content can be easily reordered or skipped altogether. Parsing pages in general becomes much less painful, which can lead to better experiences for people relying on assistive technologies. In addition, new attributes on elements can specify the roles of elements so that screenreaders can work with them easier. In Chapter 5, *Improving Accessibility*, on page 90 you'll learn how to use those new attributes so that today's screenreaders can use them.

Advanced Selectors

CSS3 has selectors that let you identify odd and even rows of tables, or all selected checkboxes, or even the last paragraph in a group. You can accomplish more with less code and less markup. This also makes it much easier to style HTML you can't edit. In Chapter 4, *Making Better User Interfaces with CSS3*, on page 68, you'll see how to use these selectors effectively.

Visual Effects

Drop shadows on text and images help bring depth to a web page, and gradients can also add dimension. CSS3 lets you add shadows and

gradients to elements without resorting to background images or extra markup. In addition, you can use transformations to round corners, or skew and rotate elements. You'll see how all of those things work in Chapter 8, *Eye Candy*, on page 123.

1.2 Backwards Compatibility

One of the best reasons for you to embrace HTML5 today is that it works in most existing browsers. Right now, even in Internet Explorer 6, you can start using HTML5 and slowly transition your markup. It'll even validate with the W3C's validation service (conditionally, of course, as the standards are still evolving.)

If you've worked with HTML or XML, you've come across the doctype declaration before. It's used to tell validators and editors what tags and attributes you can use and how the document should be formed. It's also used by a lot of web browsers to determine how the browser will render the page. A valid doctype often causes browsers to render pages in "standards mode".

Compared to the rather verbose *XHTML 1.0 Transitional* doctype used by many sites:

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

the HTML5 doctype is ridiculously simple:

[Download](#) html5_why/index.html

```
<!DOCTYPE html>
```

Place that at the top of the document and you're using HTML5.

Of course, you can't use any of the new HTML5 elements that your target browsers don't yet support, but your document will validate as HTML5.

1.3 The Road To The Future is Bumpy

There are a few roadblocks that continue to impede the widespread adoption of HTML5 and CSS3. Some are obvious, and some are less so.

Internet Explorer

Internet Explorer currently has the largest user base and has very weak HTML5 and CSS3 support. IE 9 promises to improve this situation.



Joe Asks...

But I like my XHTML self-closing tags. Can I still use those?

You sure can! Many developers fell in love with XHTML because of the stricter requirements on markup. XHTML documents forced quoted attributes, made you self-close content tags, required that you use lower-case attribute names, and brought well-formed markup onto the World Wide Web. Moving to HTML5 doesn't mean you have to change your ways. HTML5 documents will be valid if you use the HTML5-style syntax or the XHTML syntax, but you need to understand the implications of using self-closing tags.

Most web servers serve HTML pages with the text/html Mime type due to Internet Explorer's inability to properly handle the application/xml+xhtml Mime type associated with XHTML pages. Because of this, browsers tend to strip off self-closing tags because self-closing tags were not considered valid HTML before HTML5. For example, if you had a self-closing script tag above a div like this:

```
<script language="javascript" src="application.js" />
<h2>He1p</h2>
```

the browser would remove the self closing forward slash, and then the renderer would think that the h2 was *within* the script tag, *which never closes!*. This is why you see script tags coded with an explicit closing tag, even though a self closing tag is valid XHTML markup.

So, be aware of possible issues like this if you do use self-closing tags in your HTML5 documents, as they will be served with the text/html MIME type. You can learn more about this issue and others at <http://www.webdevout.net/articles/beware-of-xhtml#myths>.

Cake and Frosting

I like cake. I like pie better, but cake is pretty good stuff. I prefer cake with frosting on it.

When you're developing web applications, you have to keep in mind that all the pretty user interfaces and fancy JavaScript stuff is the frosting on the cake. Your web site can be really good without that stuff, and just like a cake, you have to have a foundation on which to put your frosting.

I've met some people who don't like frosting. They scrape it off the cake. I've also met people who use web applications without JavaScript for varying reasons.

Bake these people a really awesome cake. Then add frosting.

That doesn't mean we can't use HTML5 and CSS3 in our sites anyway. We can make our sites work in Internet Explorer, but they don't have to work the same as the versions we develop for Chrome and Firefox. We'll just provide fallback solutions so we don't anger users and lose customers.

Accessibility

Our users must be able to interact with our web sites, whether they are visually impaired, hearing impaired, on older browsers, on slow connections, or on mobile devices. HTML5 introduces some new elements, like the audio, video, and canvas. Audio and video have always had accessibility issues, but the canvas element presents new challenges. The canvas element lets you create vector images within the HTML document using JavaScript. This creates issues for the visually impaired, but also causes problems for the 5% of web users who have disabled JavaScript.³

We need to be mindful of accessibility when we push ahead with new technologies and provide suitable fallbacks for these HTML5 elements, just like we would for people using Internet Explorer.

3. <http://visualrevenue.com/blog/2007/08/eu-and-us-javascript-disabled-index.html>

Competing Corporate Interests

Internet Explorer is not the only browser slowing adoption of HTML5 and CSS3. Google, Apple, and the Mozilla Foundation each have their own agendas as well, and they're battling it out for supremacy. They're arguing over video and audio codec support, and they're including their opinions in their browser releases. For example, Safari will play MP3 audio with the audio element, but ogg files won't work. Firefox, however, supports ogg files instead of mp3 files.

Eventually these differences will be resolved. In the meantime, we can make smart choices about what we support, by limiting what we implement to our target audiences, or we can implement things multiple times, once for each browser until the standards are finalized. It's not as painful as it sounds. We'll discuss this more in Chapter 7, *Embedding Audio and Video*, on page 120.

HTML5 and CSS3 Are Still Works In Progress

They're not final specifications, and that means that anything in those specifications could change. While Firefox, Chrome, and Safari have strong HTML5 support, if the specification changes, the browsers will change with it, and this could lead to some deprecated, broken web sites. During the course of writing this book, CSS3 box shadows have been removed and re-added to the specification, and web socket protocols have been modified, breaking client-server communications entirely.

If you follow the progress of HTML5, CSS3, and stay up to date with what's happening, you'll be fine. A good portion of the things we'll be discussing in this book are going to work for a long time.

When we come across something that doesn't work in one of our target browsers, we just fill in the gaps as we go, using JavaScript and Flash as our putty. We'll build solid solutions that work for all of our users, and as time goes on, we'll be able to remove the JavaScript and other fallback solutions without changing our implementations.

But before we can think about the future, let's start working with HTML5. There are a bunch of new structural tags waiting to meet you over in the next chapter. So let's not keep them waiting, shall we?

Part I

Improving User Interfaces

New Structural Tags and Attributes

I'd like to talk to you about a serious problem affecting many web developers today. *Divitis*, a chronic syndrome that causes web developers to wrap elements with extra `div` tags with IDs like “banner”, “sidebar”, “article”, and “footer” is rampant. It's also highly contagious. Developers pass Divitis to each other extremely quickly, and since Divs are invisible to the naked eye, even mild cases of Divitis may go unnoticed for years.

Here's a common symptom of Divitis:

```
<div id="navbar_wrapper">
  <div id="navbar">
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/">Home</a></li>
    </ul>
  </div>
</div>
```

Here we have an unordered list, which is already a block element¹, wrapped with two `div` tags which are also block elements. The `id` attributes on these wrapper elements tell us what they do, but you can remove at least one of these wrappers to get the same result. Overuse of markup leads to bloat and pages that are difficult to style and maintain.

1. Remember, block elements fall on their own line, whereas inline elements do not force a line break.

There is hope though. The HTML5 specification provides a cure, in the form of new semantic tags which describe the content they contain. Together with HTML5, we can help wipe out Divitis in our lifetime.

Because so many developers have made sidebars, headers, footers, and sections in their designs, the HTML5 specification introduces new tags specifically designed to divide a page into logical regions. Let's put those new elements to work.

In addition to these new structural tags, we'll also talk about the meter element, and discuss how we can use the new custom attributes feature in HTML5 so we can embed data into our elements instead of hijacking classes or existing attributes. In a nutshell, we're going to find out how to use the right tag for the right job.

Feature	Description	Supported Browsers
header	Defines a header region of a page or section	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
footer	Defines a footer region of a page or section	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
nav	Defines a navigation region of a page or section	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
section	Defines a logical region of a page or a grouping of content	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
article	Defines an article, or complete piece of content.	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
aside	Defines secondary or related content	<ul style="list-style-type: none"> • IE 8 • Safari 4+ • Firefox 3.6+ • Chrome 5+ • Opera 10+
Custom data attributes	Allows addition of custom attributes to any	<ul style="list-style-type: none"> • All browsers support reading these

Semantic Markup

Semantic markup is all about describing your content. If you've been developing web pages for a few years, you've probably divided your pages into various regions like header, footer, and sidebar so that you could more easily identify the regions of the page when applying stylesheets and other formatting.

Semantic markup makes it easy for machines and people alike to understand the context of the content. The new HTML5 markup tags like section, header, and nav help you do just that.

1 Redefining a Blog using Semantic Markup

One place you're sure to find lots of content in need of structured markup is a blog. You're going to have headers, footers, multiple types of navigation (archives, blogrolls and internal links), and of course, articles or posts. Let's use HTML5 markup to mock up the front page of the blog for AwesomeCo, a company on the cutting edge of Awesomeness.

Take a look at Figure 2.1, on the following page to get an idea of what we're going to build. It's a fairly typical blog structure, with a main header with horizontal navigation below the header. In the main section, each article has a header and a footer. An article may also have a pull quote, or an aside. There's a sidebar which contains additional navigation elements. Finally, the page has a footer for contact and copyright information. There's nothing new about this structure, except that this time, instead of coding it up with lots of div tags, we're going to use specific tags to describe these regions.

When we're all done, we'll have something that looks like Figure 2.2, on page 25.

It All Starts With The Right Doctype

We want to use HTML5's new elements, and that means we need to let browsers and validators know about the tags we'll be using. Create a new page called blog.html and paste in a basic HTML5 template.

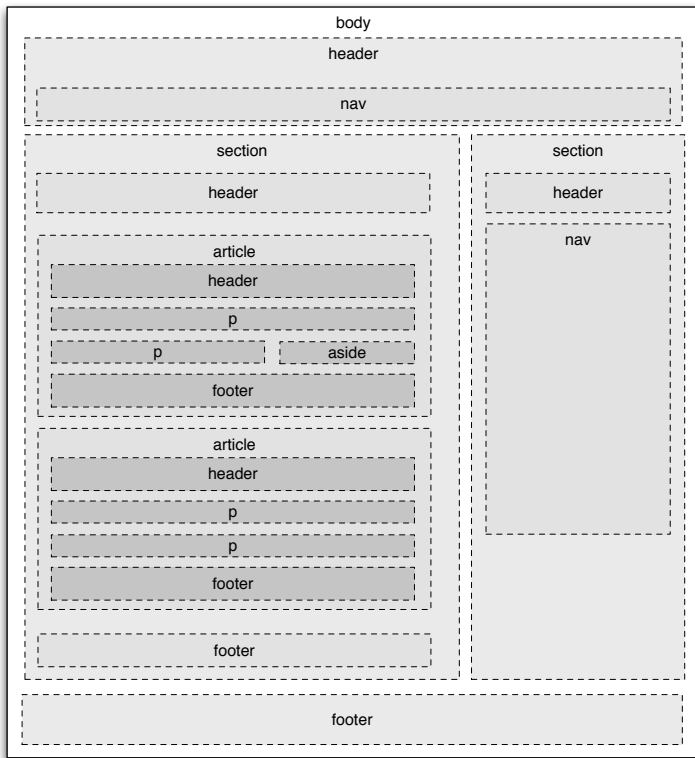


Figure 2.1: The blog structure using new HTML5 semantic markup

[Download](#) `html5newtags/index.html`

```

Line 1 <!DOCTYPE html>
2 <html lang="en-US">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5 <title>AwesomeCo Blog</title>
6 </head>
7
8 <body>
9 </body>
10 </html>

```

Take a look at the doctype on line 1 of that example. This is all we need for an HTML5 doctype. If you're used to doing web pages, you're probably familiar with the long hard-to-remember doctypes for XHTML like this:

AwesomeCo Blog!

[Latest Posts](#) [Archives](#) [Contributors](#) [Contact Us](#)

How Many Should We Put You Down For?

Posted by Brian on April 1st, 2010 at 2:39PM

The first big rule in sales is that if the person leaves empty-handed, they're likely not going to come back. That's why you have to be somewhat aggressive when you're working with a customer, but you have to make sure you don't overdo it and scare them away.

"Never give someone a chance to say no when selling your product."

One way you can keep a conversation going is to avoid asking questions that have yes or no answers. For example, if you're selling a service plan, don't ever ask "Are you interested in our 3 or 5 year service plan?" Instead, ask "Are you interested in the 3 year service plan or the 5 year plan, which is a better value?" At first glance, they appear to be asking the same thing, and while a customer can still opt out, it's harder for them to opt out of the second question because they have to say more than just "no."

[25 Comments](#) ...

© 2010 AwesomeCo.

[Home](#) [About](#) [Terms of Service](#) [Privacy](#)

Archives

- [April 2010](#)
- [March 2010](#)
- [February 2010](#)
- [January 2010](#)
- [December 2009](#)
- [November 2009](#)
- [October 2009](#)
- [September 2009](#)

Figure 2.2: The finished layout

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

Now, take another look at the HTML5 doctype:

```
<!DOCTYPE HTML>
```

Much simpler, and much easier to remember.

The point of a doctype is twofold. First, it's to help validators determine what validation rules it needs to use when validating the code. Second, a doctype forces Internet Explorer versions 6, 7, and 8 to go into "standards-mode.", which is vitally important if you're trying to build pages that work across all browsers. The HTML5 doctype satisfies both of these needs, and *is even recognized by Internet Explorer 6*.

Headers

Headers, not to be confused with headings like h1, h2 and h3, may contain all sorts of content, from the company logo to the search box. Our blog header will contain the blog's title and subtitle.

Download [html5newtags/index2.html](#)

```
Line 1 <header id="page_header">
2     <h1>AwesomeCo Blog!</h1>
3 </header>
```

You're not restricted to having just one header on a page. Each individual section or article can also have a header, and so it can be helpful to use the ID attribute like I did on 1 to uniquely identify your elements. A unique ID makes it easy to style elements with CSS or locate elements with JavaScript.

Footers

The footer element should be used to define footer information for a document or an adjacent section. You've seen footers before on web sites. They usually contain information like the copyright date and information on who owns the site. The specification says we can have multiple footers in a document too, so that means we could use the footers within our blog articles too.

For now, let's just define a simple footer for our page. Since we can have more than one footer, we'll give this one an ID just like we did with the header. It'll help us uniquely identify this particular footer when we want to add styles to this element and its children.

Download [html5newtags/index.html](#)

```
<footer id="page_footer">
  <p>&copy; 2010 AwesomeCo.</p>
</footer>
```

This footer simply contains a copyright date. However, like headers, footers on pages often contain other elements, including navigational elements.

Navigation

Navigation is vital to the success of a web site. People simply aren't going to stick around if you make it too hard for them to find what they're looking for, so it makes sense for navigation to get its own HTML tag.

Let's add a navigation section to our document's header. We'll add links to the blog's home page, the archives, a page that lists the contributors to the blog, and a link to a contact page.

Download [html5newtags/index.html](#)

```
<header id="page_header">
  <h1>AwesomeCo Blog!</h1>
  <nav>
    <ul>
      <li><a href="/">Latest Posts</a></li>
      <li><a href="archives">Archives</a></li>
      <li><a href="contributors">Contributors</a></li>
      <li><a href="contact">Contact Us</a></li>
    </ul>
  </nav>
</header>
```

Like headers and footers, your page can have multiple navigation elements. You often find navigation in your header and in your footer, and so now you can identify those explicitly. Our blog’s footer needs to have links to the AwesomeCo home page, the company’s “about us” page, and links to the company’s Terms of Service and Privacy policies. We’ll add these as another unordered list within in the page’s footer element.

Download [html5newtags/index.html](#)

```
<footer id="page_footer">
  <p>&copy; 2010 AwesomeCo.</p>
  <nav>
    <ul>
      <li><a href="http://awesomeco.com/">Home</a></li>
      <li><a href="about">About</a></li>
      <li><a href="terms.html">Terms of Service</a></li>
      <li><a href="privacy.html">Privacy</a></li>
    </ul>
  </nav>
</footer>
```

We will use CSS to change how both of these navigation bars look, so don’t worry too much about the appearance yet. The point of these new elements is to describe the content, not to describe how the content looks.

Sections and Articles

Sections are the logical regions of a page, and the section element is here to replace the abused div tag.

Download [html5newtags/index.html](#)

```
<section id="posts">
</section>
```

Don't get carried away with sections though. Use them to logically group your content! I created a section that will hold all of the blog posts. However, each post shouldn't be in its own section. We have a more appropriate tag for that.

Articles

The article tag is the perfect element to describe the actual content of a web page. With so many elements on a page, including headers, footers, navigational elements, advertisements, widgets, blogrolls, and social media bookmarks, it might be easy to forget that people come to your site because they're interested in the content you're providing. The article tag helps you describe that content.

[Download](#) `html5newtags/index.html`

```
<article class="post">
  <p>
    The first big rule in sales is that if the person leaves empty-handed,
    they're likely not going to come back. That's why you have to be
    somewhat aggressive when you're working with a customer, but you have
    to make sure you don't overdo it and scare them away.
  </p>
  <p>
    One way you can keep a conversation going is to avoid asking questions
    that have yes or no answers. For example, if you're selling a service
    plan, don't ever ask "Are you interested in our 3 or 5 year
    service plan?" Instead, ask "Are you interested in the 3
    year service plan or the 5 year plan, which is a better value?"
    At first glance, they appear to be asking the same thing, and while
    a customer can still opt out, it's harder for them to opt out of
    the second question because they have to say more than just
    "no."
  </p>
</article>
```

Each of our articles will have a header, some content, and a footer, like this:

[Download](#) `html5newtags/index.html`

```
<article class="post">
  <header>
    <h2>How Many Should We Put You Down For?</h2>
    <p>Posted by Brian on
      <time datetime="2010-04-01T14:39">April 1st, 2010 at 2:39PM</time>
    </p>
  </header>
  <p>
    The first big rule in sales is that if the person leaves empty-handed,
    they're likely not going to come back. That's why you have to be
```

somewhat aggressive when you're working with a customer, but you have to make sure you don't overdo it and scare them away.

```

</p>
<p>
One way you can keep a conversation going is to avoid asking questions
that have yes or no answers. For example, if you're selling a service
plan, don't ever ask &quot;Are you interested in our 3 or 5 year
service plan?&quot; Instead, ask &quot;Are you interested in the 3
year service plan or the 5 year plan, which is a better value?&quot;
At first glance, they appear to be asking the same thing, and while
a customer can still opt out, it's harder for them to opt out of
the second question because they have to say more than just
&quot;no.&quot;
</p>
<footer>
  <p><a href="comments"><i>25 Comments</i></a> ...</p>
</footer>
</article>

```

Asides and Sidebars

Sometimes you have content that adds something extra to your main content, like pullout quotes, diagrams, additional thoughts, or related links. You can use the new `aside` tag to identify these elements.

[Download](#) `html5newtags/index.html`

```

<aside>
  <p>
    &quot;Never give someone a chance to say no when
    selling your product.&quot;
  </p>
</aside>

```

We'll place the callout quote in an aside element. We'll nest this aside within the article, keeping it close to its related content.

[Download](#) `html5newtags/index.html`

```

<section id="posts">
  <article class="post">
    <header>
      <h2>How Many Should We Put You Down For?</h2>
      <p>Posted by Brian on
        <time datetime="2010-04-01T14:39">April 1st, 2010 at 2:39PM</time>
      </p>
    </header>

    <aside>
      <p>
        &quot;Never give someone a chance to say no when
        selling your product.&quot;
      </p>
    </aside>
  </article>
</section>

```

```

    </p>
  </aside>
  <p>
    The first big rule in sales is that if the person leaves empty-handed,
    they're likely not going to come back. That's why you have to be
    somewhat aggressive when you're working with a customer, but you have
    to make sure you don't overdo it and scare them away.
  </p>
  <p>
    One way you can keep a conversation going is to avoid asking questions
    that have yes or no answers. For example, if you're selling a service
    plan, don't ever ask &quot;Are you interested in our 3 or 5 year
    service plan?&quot; Instead, ask &quot;Are you interested in the 3
    year service plan or the 5 year plan, which is a better value?&quot;
    At first glance, they appear to be asking the same thing, and while
    a customer can still opt out, it's harder for them to opt out of
    the second question because they have to say more than just
    &quot;no.&quot;
  </p>
  <footer>
    <p><a href="comments"><i>25 Comments</i></a> ...</p>
  </footer>
</article>
</section>

```

Now we just have to add the sidebar section.

Asides are not Page Sidebars!

Our blog has a sidebar on the right side that contains links to the archives for the blog. If you're thinking that we could use the `aside` tag to define the sidebar of our blog, you'd be wrong. You *could* do it that way, but it goes against the spirit of the specification. The `aside` is designed to show content related to an article. It's a good place to show related links, a glossary, or a pullout quote.

To mark up our sidebar that contains our list of prior archives, we'll just use another section tag and a `nav` tag.

[Download](#) html5newtags/index.html

```

<section id="sidebar">
  <nav>
    <h3>Archives</h3>
    <ul>
      <li><a href="2010/06">June 2010</a></li>
      <li><a href="2010/05">May 2010</a></li>
      <li><a href="2010/04">April 2010</a></li>
      <li><a href="2010/03">March 2010</a></li>
      <li><a href="2010/02">February 2010</a></li>
      <li><a href="2010/01">January 2010</a></li>
    </ul>
  </nav>
</section>

```

```

    <li><a href="2009/12">December 2009</a></li>
  </ul>
</nav>
</section>

```

That's it for our blog's structure. Now we can start applying styles to these new elements.

Styling

Each of these new elements can be styled just like you'd style div tags. Let's first center the page's content and set some basic font styles.

[Download](#) html5newtags/style.css

```

body{
  width:960px;
  margin:15px auto;
  font-family: Arial, "MS Trebuchet", sans-serif;
}

p{
  margin:0 0 20px 0;
}

p, li{
  line-height:20px;
}

```

Next, we define the header's width.

[Download](#) html5newtags/style.css

```

header#page_header{
  width:100%;
}

```

We style the navigation links by transforming the bulleted lists into horizontal navigation bars.

[Download](#) html5newtags/style.css

```

header#page_header nav ul, #page_footer nav ul{
  list-style: none;
  margin: 0;
  padding: 0;
}
#page_header nav ul li, footer#page_footer nav ul li{
  padding:0;
  margin: 0 20px 0 0;
  display:inline;
}

```

The posts section needs to be floated left and given a width, and we also need to float the callout inside of the article. While we're doing that, let's bump up the font size for the callout.

[Download](#) `html5newtags/style.css`

```
section#posts{
  float: left;
  width: 74%;
}

section#posts aside{
  float: right;
  width: 35%;
  margin-left: 5%;
  font-size: 20px;
  line-height: 40px;
}
```

We'll also need to float the sidebar and define its width.

[Download](#) `html5newtags/style.css`

```
section#sidebar{
  float: left;
  width: 25%;
}
```

And we need to define the footer. We'll clear the floats on the footer so that it sits at the bottom of the page.

[Download](#) `html5newtags/style.css`

```
footer#page_footer{
  clear: both;
  width: 100%;
  display: block;
  text-align: center;
}
```

These are just basic styles. From here, I'm confident you can make this look much, much better.

Falling Back

While this all works great in Firefox, Chrome, and Safari, the people in management aren't going to be too happy when they see the mess that Internet Explorer makes out of our page. The content displays fine, but since IE doesn't understand these elements, it can't apply styles to them, and the whole page resembles something from the mid 1990s.

The only way to make IE style these elements is to use JavaScript to define the elements as part of the document. That turns out to be really easy. We'll add this code to our head section of the page so it executes before the browser renders any elements:

[Download](#) `html5newtags/index.html`

```
<script type="text/javascript">
  document.createElement("nav");
  document.createElement("header");
  document.createElement("footer");
  document.createElement("section");
  document.createElement("aside");
  document.createElement("article");
</script>
```

Internet Explorer works just great after you define each of the new elements. You are creating a dependency on JavaScript though, so you need to take that into consideration. The improved organization and readability of the document make it tempting, and since there are no accessibility concerns, as the contents still display and are read by a screen reader, you're only making the presentation seem grossly out of date to your users who have disabled JavaScript intentionally.

2

Showing Progress with the Meter Element

AwesomeCo is holding a charity fundraiser in a few months and they're looking to get \$5000 donated by the general public. Because they're such an awesome company, they're planning to kick in an additional \$5000 if people pledge enough support to hit the original \$5000 goal. AwesomeCo wants to display a progress meter on one of their pages. When we're done, we'll have something that looks like Figure 2.3, on the following page .

While we can certainly achieve that with some divs styled with CSS, we can also use the new meter element which is designed specifically for this task.

The meter element helps you semantically describe an actual meter. In order for your meter to be in harmony with the specification, you can't use your meter for things with arbitrary minimum or maximum values like height and weight or temperatures because there's no technical minimum and maximum value for those. ² In our case, we want to show how close we are to our goal of \$5000. We have a minimum and a maximum value so it's a perfect fit for us.

We represent our meter with this code (we'll hard-code \$2500 as our current value just to demonstrate how it works right now.)

[Download html5_meter/index.html](#)

```
<section id="pledge">
  <header>
    <h3>Our Fundraising Goal</h3>
  </header>
  <meter title="USD" id="pledge_goal" value="2500" min="0" max="5000">
    $2500.00
  </meter>
  <p>Help us reach our goal of $5000!</p>
</section>
```

We're making use of our new structural elements here too. The meter tag doesn't have any default styling, so when we look at our page in our

2. You could use the meter element for temperature if you treated it like a *thermometer*, where you would set low and high values.



Figure 2.3: A meter to show progress toward our goal

browser, we won't see anything other than the text we placed within the tag itself. We can style this with a little bit of CSS, but since we're looking to make this work across all browsers, we'll use some jQuery as well. First, let's define some basic CSS styles.

Defining Styles for the Meter

Our meter is going to consist of an outer box which will represent the total length of the meter, an inner box, which we'll call the "fill", and the text label itself that shows the dollar amount. The styling for the meter itself is simple.

[Download](#) `html5_meter/index.html`

```
meter{
  width: 280px;
  display: block;
  border: 1px solid #000;
  position: relative;
}
```

We make it a block-level element, set some padding, and give it a border. Then we define the inner fill with a gradient.

[Download](#) `html5_meter/index.html`

```
.fill{
  background-color: #999;
  background-image: -webkit-gradient(
```

```

    linear,
    left bottom,
    left top,
    color-stop(0.37, rgb(14,242,30)),
    color-stop(0.69, rgb(41,255,57))
);

background-image: -moz-linear-gradient(
    center bottom,
    rgb(14,242,30) 37%,
    rgb(41,255,57) 69%
);
}

```

That gradient syntax is a little complex, but we'll talk about it in more detail in Chapter 8, *Eye Candy*, on page 123. Once we've styled the fill, we need to place the existing label inside of the bar.

[Download](#) `html5_meter/index.html`

```

.label{
    position: absolute;
    top: 0;
    right: 0;
    z-index: 1000;
}

```

We're absolutely positioning the label inside of the bar. It will sit on a layer above the fill.

Notice that we've defined these as classes, and yet we don't have any classes for the fill or the label in our markup. That's because the meter doesn't have any default presentation at all, so we'll be using JavaScript to add some elements inside of the meter. We'll assign classes to the elements we create.

Building the Meter with jQuery

The meter element isn't recognized by every browser, and neither are the section and header elements we're using, so we'll enable support for those with our JavaScript hack from Section 1, *Falling Back*, on page 32.

[Download](#) `html5_meter/index.html`

```

// IE support
document.createElement("meter");
document.createElement("section");
document.createElement("header");

```

Then we turn the meter into a structure we can visualize by appending some elements within the meter and applying our styles we defined by applying classes to our elements.

[Download](#) html5_meter/index.html

```

Line 1 $(function(){
-     var meter = $("#pledge_goal");
-
-     var label = $("<span>" + meter.html() + "</span>");
5     label.addClass("label");
-
-     var fill = $("<div></div>");
-     fill.addClass("fill");
-     fill.css("width", (meter.attr("value") / meter.attr("max") * 100) + "%");
10    fill.append("<div style='clear:both;'><br></div>");
-
-     meter.html("");
-     meter.append(fill);
-     meter.append(label);
15 });

```

On 9 we compute the width of the fill area with some simple arithmetic using numbers we grab out of the meter element. Notice on line 12 we remove the original HTML content, since we're duplicating it on 4.

With the JavaScript in place, our meter is finished and ready for us to show off to the world. When we bring this live, we'll just replace the hard-coded value inside the meter element with a value we populate from our database when we render the page. The JavaScript code will then show the updated value each time a user refreshes the page.

Falling Back

The meter element doesn't have any default style, and the way we've implemented it here works in IE 6, 7, and 8. We're using the new meter element to semantically mark up the information about our pledge, just like the way we handled the sections of our blog in *Redefining a Blog using Semantic Markup*, on page 23.

Users without JavaScript enabled will still see the content we placed within the opening and closing meter element, which means we have to be mindful of what we place there.

Progress bars

The HTML5 specification also mentions an element called `progress` which is designed to show progress toward a goal. It's very similar to a meter but it's designed to show active progress like you'd see if you were uploading a file. A meter, by comparison, is designed to show a measurement that's not currently moving, like a snapshot of available storage space on the server for a given user. The markup for a progress bar is very similar to the meter element.

[Download](#) `html5_meter/progress.html`

```
<progress id="progressbar" max=100><span>0</span>%</progress>
```

The `progress` element isn't officially supported by any browsers yet, but you can use jQuery to manipulate and style the element just like we did for meter.

3

Creating Popup Windows with Custom Data Attributes

If you've built any web application that uses JavaScript to grab information out of the document, you know that it can sometimes involve a bit of hackery and parsing to make things work. You'll end up inserting extra information into event handlers or abusing the `rel` or `class` attributes to inject behavior. Those days are now over thanks to the introduction of custom data attributes.

Custom data attributes all start with the prefix `data-` and are ignored by the validator for HTML5 documents. You can attach a custom data attribute to any element you'd like, whether it be metadata about a photograph, latitude and longitude coordinates, or, as you'll see in this section, dimensions for a popup window.

The biggest advantage of custom data attributes is that you can use them right now in nearly every web browser, since they can be easily grabbed with JavaScript.

Over the years, popup windows have gotten a bad rap, and often rightly so. They're often used to get you to look at an ad, convince unsuspecting

web surfers to install spyware or viruses, or worse, to give away personal information which is then resold. It's no wonder most browsers have some type of popup-blocker available.

Popups aren't all bad though. Web application developers often rely on popup windows to display online help, additional options, or other important user interface features. To make popups less annoying, we need to implement them in an unobtrusive manner.

Separating Behavior from Content, or why onclick is bad

When you look at AwesomeCo's human resources page, you see several links that display policies in popup windows. Most of them look like this:

[Download](#) `html5_popups_with_custom_data/original_example_1.html`

```
<a href='#'
  onclick="window.open('holiday_pay.html',WinName,'width=300,height=300');">
  Holiday pay
</a>
```

This is a pretty common way to build links that spawn popups. In fact, this is the way JavaScript newbies often learn how to make popup windows. There are a couple of problems that we should address with this approach before moving on, though.

Improve Accessibility

The link destination isn't set! If JavaScript is disabled, the link won't take the user to the page. That's a huge problem we need to address immediately. Do not *ever* omit the href attribute or give it a value like this under *any* circumstances. Give it the address of the resource that would normally pop up.

[Download](#) `html5_popups_with_custom_data/original_example_2.html`

```
<a href='holiday_pay.html'
  onclick="window.open(this.href,WinName,'width=300,height=300');">
  Holiday pay
</a>
```

The JavaScript code then reads the attached element's href attribute for the link's location.

The first step towards building accessible pages is to ensure that all of the functionality works *without* JavaScript.

Abolish The onclick

Keep the behavior separate from the content, just like you keep the presentation information separate by using linked stylesheets. Using onclick is easy at first, but imagine a page with fifty links, and you'll see how the onclick method gets out of hand. You'll be repeating that JavaScript over and over again. And if you generate this code from some server-side code, you're just increasing the number of JavaScript events and making the resulting HTML much bigger than it needs to be.

Instead, give each of the anchors on the page a class that identifies them.

[Download](#) html5_popups_with_custom_data/original_example_3.html

```
<a href="holiday_pay" class="popup">Holiday Pay</a>
```

[Download](#) html5_popups_with_custom_data/original_example_3.html

```
var links = $("a.popup");

links.click(function(event){
    event.preventDefault();
    window.open($(this).attr('href'));
});
```

We use a jQuery selector to grab the element with the class of popup and then we add an observer to each element's click event. The code we pass to the click method will be executed when someone clicks the link. The preventDefault method prevents the default click event behavior. In this case, it prevents the browser from following the link and displaying a new page.

One thing we've lost though is the information on how to size and position the window, which is something we had in the original example. We want a page designer who isn't that familiar with JavaScript to still be able to set the dimensions of a window on a per-link basis.

Custom Data Attributes To The Rescue!

Situations like this are so common when building any JavaScript-enabled application. As we've seen, storing the window's desired height and width with the code is desirable, but the onclick approach has lots of drawbacks. What we can do instead is embed these attributes as attributes on the element. All we have to do is construct the link like this:

A word of caution

In this example, we used custom data attributes to provide additional information to a client-side script. It's a clever approach to a specific problem and illustrates one way to use these attributes. However, it does tend to mix presentation information with our markup. However, it's a simple way to show you how easy it is to use JavaScript to read values you embed in your page.

[Download](#) `html5_popups_with_custom_data/popup.html`

```
<a href="help/holiday_pay.html"
  data-width="600"
  data-height="400"
  title="Holiday Pay"
  class="popup">Holiday pay</a>
```

Now we just modify the click event we wrote to grab the options from the custom data attributes of the link and pass them to the `window.open` method.

[Download](#) `html5_popups_with_custom_data/popup.html`

```
$(function(){
  $(".popup").click(function(event){
    event.preventDefault();
    var href = $(this).attr("href");
    var width = $(this).attr("data-width");
    var height = $(this).attr("data-height");
    var popup = window.open (href, "popup",
      "height=" + height + ",width=" + width + "");
  });
});
```

That's all there is to it! The link now opens in a new window.

Falling Back

These attributes work in older browsers right now as long as they support JavaScript. The custom data attributes won't trip up the browser, and your document will be valid since you're using the HTML5 doctype, since the attributes that start with `data-` will all be ignored.

The Future

We can do some interesting things with these new tags and attributes once they're widely supported. We can identify and disable navigation and article footers very easily using print stylesheets.

```
nav, article>footer{display:none}
```

We can use scripting languages to quickly identify all of the articles on a page, or on a site. But most importantly, we mark up content with appropriate tags that describe it so we can write better stylesheets and better JavaScript.

Custom data attributes give developers the flexibility to embed all sorts of information in their markup. In fact, we'll use them again in Chapter 6, *Drawing On The Canvas*, on page 104. You can use them with JavaScript to determine whether or not a form tag should submit via AJAX, by simply locating any form tag with `data-remote=true`, which is something that the Ruby on Rails framework is doing. You can also use them to display dates and times in the users' time zone while still caching the page. Simply put the date on the HTML page as UTC and convert it to the users' local time on the client-side. These attributes allow you to embed real, usable data in your pages, and you can expect to see more and more frameworks and libraries taking advantage of them. I'm sure you'll find lots of great uses for them in your own work.

And we can help wipe out Divitis once and for all!

Creating User-friendly Web Forms

If you've ever designed a complicated user interface, you know how limiting the basic HTML form controls are. You're stuck using text fields, select menus, radio buttons, and checkboxes, and sometimes the even clunkier *multiple select* lists which you constantly have to explain to your users how to use ("Hold down the control key and click the entries you want, unless you're on a Mac, in which case use the Command key.")

So, you do what all good web developers do—you turn to Prototype, jQuery, or you roll your own controls and features using a combination of HTML, CSS, and JavaScript. But when you look at a form that has sliders, calendar controls, spinboxes, auto-complete fields, and visual editors, you quickly realize that you've created a nightmare for yourself. You'll have to make sure that none of the controls you include on your page conflict with any of the other controls you've included, or any of the other JavaScript libraries on the page. You can spend hours implementing a calendar picker only to find out later that now the Prototype library is having problems because jQuery took over the `$()` function. So you use jQuery's `noConflict()` method, but then you find out that the color picker control you used no longer works because that plugin wasn't written carefully enough.

If you're smiling, it's because you've been there. If you're fuming, I'm guessing it's for the same reason. There is hope though. In this chapter we're going to build a couple of web forms using some new form field types, and we'll also implement autofocusing and placeholder text. Finally, we'll discuss how to use the new `contenteditable` attribute to

turn any HTML field into a user input control. Let's start by learning about some of the extremely useful field types.

Feature	Use	Supported Browsers
Email field	<code><input type="email"></code>	<ul style="list-style-type: none"> • Opera 10 • iPhone, iPod Touch, iPad
URL field	<code><input type="url"></code>	<ul style="list-style-type: none"> • Opera 10 • iPhone, iPod Touch, iPad
Telephone field	<code><input type="tel"></code>	<ul style="list-style-type: none"> • Opera 10
Search field	<code><input type="search"></code>	<ul style="list-style-type: none"> • Opera 10 • Safari 4 • Chrome 5 • iPhone, iPod Touch, iPad
Slider controls	<code><input type="range"></code>	<ul style="list-style-type: none"> • Safari 4 • Chrome 5 • Opera 10
Number fields	<code><input type="number"></code>	<ul style="list-style-type: none"> • Opera 10 • Safari 5 • iPhone, iPod Touch, iPad
Calendar controls	<code><input type="date"></code> <code><input type="month"></code> <code><input type="week"></code>	<ul style="list-style-type: none"> • Opera 10
Calendars with Time	<code><input type="datetime"></code> <code><input type="time"></code>	<ul style="list-style-type: none"> • Opera 10
Color fields	<code><input type="color"></code>	<ul style="list-style-type: none"> • Chrome 5¹
Autofocus	<code><input type="text" autofocus="true"></code>	<ul style="list-style-type: none"> • Safari 4 • Chrome 5 • Firefox 3.6

4

Describing Data with New Input Fields

HTML5 introduces several new input types that you can use to better describe the type of data your users are entering. In addition to the standard text fields, radio buttons, and checkbox elements, you can use elements like email fields, calendars, color pickers, spinboxes, and sliders. You can see a complete list in Figure 3.1, on the previous page. Browsers can use these new fields to display better controls to the user without the need for JavaScript. Mobile devices and virtual keyboards for tablets and touchscreens can use the field types to display different keyboard layouts. For example, the iPhone's Mobile Safari browser displays alternate keyboard layouts when the user is entering data into the URL and email types, making special characters like @, ., :, and / easily accessible.

Improving the AwesomeCo Projects Form

AwesomeCo is working on creating a new project management web application to make it easier for developers and managers to keep up with the progress of the many projects they have going on. Each project has a name, a contact email address, and a staging URL so managers can preview the web site as it's being built. There are also fields for the start date, priority, and estimated number of hours the project should take to complete. Finally, the development manager would like to give each project a color so he can quickly identify each project when he looks at reports.

Let's mock up a quick project preferences page using the new HTML5 fields.

Setting up the Basic Form

Let's create a basic HTML form that does a POST request. Since there's nothing special about the name field, we'll use the trusty text field.

[Download](#) `html5forms/index.html`

```
<form method="post" action="/projects/1">

  <fieldset id="personal_information">
    <legend>Project Information</legend>
    <ol>
```

```

    <li>
      <label for="name">Name</label>
      <input type="text" name="name" autofocus="true" id="name">
    </li>

  </ol>

</fieldset>

</form>

```

Notice that we're marking this form up with labels wrapped in an ordered list. Labels are an important part of accessibility. The `for` attribute of the label references the `id` of its associated form element. This helps screen readers identify fields on a page. The ordered list provides a good way of listing the fields without resorting to complex table or `div` structures. This also gives you a way to mark up the order in which you'd like people to fill out the fields.

Creating a Slider Using Range

Sliders are commonly used to let users decrease or increase a numerical value, and could be a great way to quickly allow managers to both visualize and modify the priority of the project. You implement a slider with the `range` type.

[Download](#) `html5forms/index.html`

```

<label for="priority">Priority</label>
<input type="range" min="0" max="10" name="priority" value="0" id="priority">

```

Add this to the form, within a new `li` element just like the previous field.

Chrome and Opera both implement a Slider widget, which looks like this:

Priority



Notice that we've also set the `min` and `max` range for the slider. That will constrain the value of the form field.

Handling Numbers with Spinboxes

We use numbers a lot, and while typing numbers is fairly simple, spinboxes can make making minor adjustments easier. A spinbox is a control with arrows that increment or decrement the value in the box. Let's use the spinbox for estimated hours. That way the hours can be easily adjusted.

[Download](#) `html5forms/index.html`

```
<label for="estimated_hours">Estimated Hours</label>
<input type="number" name="estimated_hours"
       min="0"
       id="estimated_hours">
```

Opera supports the spinbox control, which looks like this:

Estimated Hours

The spinbox also allows typing by default, and like range sliders, we can set minimum and maximum values. However, those minimum and maximum ranges won't be applied to any value you type into the field.

Also notice that you can control the size of the increment step by giving a value to the `step` parameter. It defaults to 1, but can be any numerical value.


Dates

Recording the start date of the project is pretty important, and we want to make that as easy as possible. The `date` input type is a perfect fit here.

[Download](#) `html5forms/index.html`

```
<label for="start_date">Start date</label>
<input type="date" name="start_date" id="start_date">
```

Opera is the only browser that currently supports the calendar picker. Here's an example of their implementation:

<p>Email contact</p> <p>Staging URL</p> <p> Project color</p>	<div style="display: flex; justify-content: space-between; align-items: center;"> ◀ March ▶ 2010 ⬆ </div> <table border="1" style="width: 100%; text-align: center;"> <thead> <tr> <th>Week</th> <th>Mon</th> <th>Tue</th> <th>Wed</th> <th>Thu</th> <th>Fri</th> <th>Sat</th> <th>Sun</th> </tr> </thead> <tbody> <tr> <td>9</td> <td><u>1</u></td> <td><u>2</u></td> <td><u>3</u></td> <td><u>4</u></td> <td><u>5</u></td> <td><u>6</u></td> <td><u>7</u></td> </tr> <tr> <td>10</td> <td><u>8</u></td> <td><u>9</u></td> <td><u>10</u></td> <td><u>11</u></td> <td><u>12</u></td> <td><u>13</u></td> <td><u>14</u></td> </tr> <tr> <td>11</td> <td><u>15</u></td> <td><u>16</u></td> <td><u>17</u></td> <td><u>18</u></td> <td><u>19</u></td> <td><u>20</u></td> <td><u>21</u></td> </tr> <tr> <td>12</td> <td><u>22</u></td> <td><u>23</u></td> <td><u>24</u></td> <td><u>25</u></td> <td><u>26</u></td> <td><u>27</u></td> <td><u>28</u></td> </tr> <tr> <td>13</td> <td><u>29</u></td> <td><u>30</u></td> <td><u>31</u></td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>14</td> <td>5</td> <td>6</td> <td>7</td> <td>8</td> <td>9</td> <td>10</td> <td>11</td> </tr> </tbody> </table> <div style="display: flex; justify-content: space-between; margin-top: 5px;"> Today None </div>							Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun	9	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	10	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>	11	<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	<u>19</u>	<u>20</u>	<u>21</u>	12	<u>22</u>	<u>23</u>	<u>24</u>	<u>25</u>	<u>26</u>	<u>27</u>	<u>28</u>	13	<u>29</u>	<u>30</u>	<u>31</u>	1	2	3	4	14	5	6	7	8	9	10	11
	Week	Mon	Tue	Wed	Thu	Fri	Sat	Sun																																																							
	9	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>																																																							
	10	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>																																																							
	11	<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	<u>19</u>	<u>20</u>	<u>21</u>																																																							
	12	<u>22</u>	<u>23</u>	<u>24</u>	<u>25</u>	<u>26</u>	<u>27</u>	<u>28</u>																																																							
	13	<u>29</u>	<u>30</u>	<u>31</u>	1	2	3	4																																																							
	14	5	6	7	8	9	10	11																																																							

Other browsers render a text field.

Email

The HTML5 specification says that the Email input type is designed to hold either a single email address or an email address list, so that's the perfect candidate for our email field.

[Download](#) [html5forms/index.html](#)

```
<label for="email">Email contact</label>
<input type="email" name="email" id="email">
```

Most browsers render this as a text field, but Opera adds a little icon within the field to help identify it more easily.

Email contact



Mobile devices get the most benefit from this type of form field, as the virtual keyboard layouts change to make entering email addresses easier.

URL

There's a field type designed to handle URLs too. This one is especially nice if your visitor uses an iPhone, because it displays a much different keyboard layout, displaying helper buttons for quickly entering web addresses, similar to the keyboard displayed when entering a URL into Mobile Safari's address bar. Adding the staging URL field is as simple as adding this code:

[Download](#) [html5forms/index.html](#)

```
<label for="url">Staging URL</label>
<input type="url" name="url" id="url">
```

Like the Email field, Opera denotes the URL field with an icon.

Staging URL



Virtual keyboards use this field type to display a different layout as well.

Color

Finally, we need to provide a way to enter a color code, and we'll use the color type for that.

[Download](#) [html5forms/index.html](#)

```
<label for="project_color">Project color</label>
<input type="color" name="project_color" id="project_color">
```

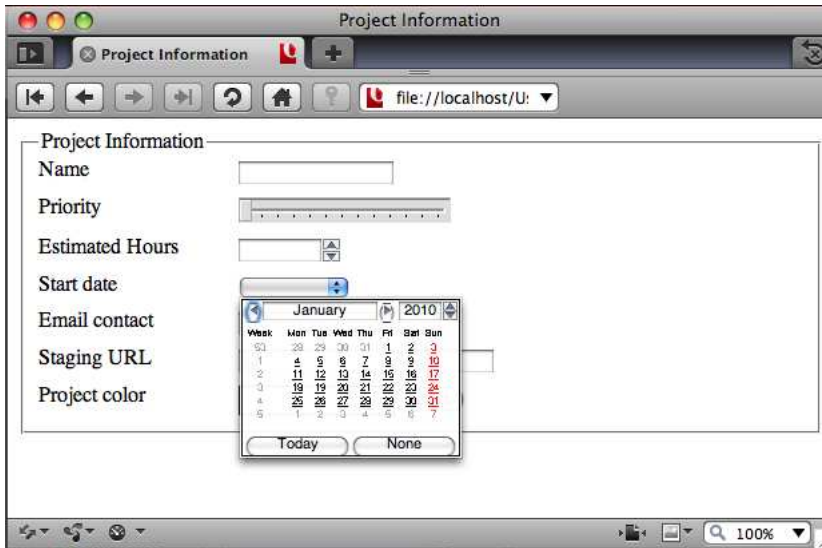


Figure 3.2: Some form controls are already supported in Opera

At the time of writing, no browsers properly display a colorpicker, but that shouldn't stop you from using this field. You're using proper markup to describe your content, and that's going to come in handy in the future, especially when you need to provide fallback support.

Opera supports most of these new controls right now, as you can see in Figure 3.2, but when you open the page in Firefox, Safari, or Google Chrome, you won't see much of a difference. We'll need to fix that.

Falling Back

Browsers that don't understand these new types simply fall back to the text type, so your forms will still be usable. At that point, you can bind one of the jQuery UI or YUI widgets to that field to transform it. As time goes on, and more browsers support these controls, you can remove the JavaScript hooks.

Replacing the Color Picker

We can easily identify and replace the color picker using jQuery with CSS3's attribute selectors. We locate any input field with the type of "color" and apply a jQuery plugin called SimpleColor.

[Download](#) html5forms/index.html

```
if (!hasColorSupport()){
  $('input[type=color]').simpleColor();
}
```

Since we used the new form types in our markup, we don't have to add an additional class name or other markup to identify the color pickers. Attribute selectors and HTML5 go together quite well.

We don't want to use this colorpicker plugin if the browser has native support for it, so we'll use some JavaScript to detect if the browser supports input fields with a type of color.

[Download](#) html5forms/index.html

```
Line 1 function hasColorSupport(){
-   input = document.createElement("input");
-   input.setAttribute("type", "color");
-   var hasColorType = (input.type !== "text");
5   // handle Safari/Chrome partial implementation
-   if(hasColorType){
-     var testString = "foo";
-     input.value=testString;
-     hasColorType = (input.value != testString);
10  }
-   return(hasColorType);
- }
```

First, we use plain JavaScript to create an element and set its type attribute to color. Then we retrieve the type attribute to see if the browser allowed us to set the attribute. If it comes back with a value of color then we have support for that type. If not, we'll have to apply our script.

Things get interesting on line 6. Safari 5 and Google Chrome 5 have partially implemented the color type. They support the field but they don't actually display a color widget. We still end up with a text field on the page. So, in our detection method, we set the value for our input field and see if the value sticks around. If it doesn't, we can assume that the browser has implemented a color picker because the input field isn't acting like a text box..

The whole bit of code to replace the color picker looks like this:

[Download](#) html5forms/index.html

```
if (!hasColorSupport()){
  $('input[type=color]').simpleColor();
}
```

That solution works, but it's very brittle. It targets a specific set of browsers, and only for the color control. Other controls have their own quirks that you need to learn. Thankfully there's an alternative solution.

Modernizr

The Modernizr² library can detect support for many HTML5 and CSS3 features. It doesn't add the missing functionality, but it does provide several mechanisms similar to the solution we implemented for detecting form fields that are more bulletproof.

Before you start throwing Modernizr in your projects, be sure you take some time to understand how it works. Whether you wrote the code yourself or not, if you use it in your project, you're responsible for it. Modernizr wasn't ready to handle Safari's partial support of the color field right away. When the next version of Chrome or Firefox comes out, you may have to hack together a solution. Who knows, maybe you'll be able to contribute that solution back to Modernizr!

You'll implement fallbacks for controls like the Date picker and the Slider in the same manner. Sliders and date pickers are included as components in the jQuery UI library.³ You'll include the jQuery UI library on the page, detect if the browser supports the control natively, and if it doesn't, apply the JavaScript version instead. Eventually you'll be able to phase out the JavaScript controls and rely completely on the controls in the browser. Because of the complexity involved with detecting these types, Modernizer will be very helpful to you. However, we'll continue writing our own detection techniques throughout the rest of this book so you can see how they work.

Aside from new form field types, HTML5 introduces a few other attributes for form fields that can help improve usability. Let's take a look at out-of-focus next.

2. <http://www.modernizr.com/>

3. <http://jqueryui.com/>

5

Jumping to the First Field with Autofocus

You can really speed up data-entry if you set the user's focus to the first field on the form when they load the page. Many search engines do this, and now HTML5 provides this capability as part of the language.

All you have to do is add `autofocus="true"` to any form field, like we already did on the profile page we built in *Describing Data with New Input Fields*, on page 46.

[Download](#) `html5forms/index.html`

```
<label for="name">Name</label>
<input type="text" name="name" autofocus="true" id="name">
```

You can only have one autofocus attribute on a page for it to work reliably. If you have more than one, the browser will focus the user's cursor onto the last autofocused form field.

Falling Back

We can detect the presence of the autofocus attribute with a little bit of JavaScript, and then use jQuery to focus on the element when the user's browser doesn't have autofocus support. This is probably the easiest fallback solution you'll come across.

[Download](#) `html5forms/autofocus.js`

```
function hasAutofocus() {
  var element = document.createElement('input');
  return 'autofocus' in element;
}

$(function(){
  if(!hasAutofocus()){
    $('input[autofocus=true]').focus();
  }
});
```

Just include this JavaScript on your page and you'll have autofocus support where you need it.

Autofocus makes it a little easier for users to start working with your forms when they load, but you may want to give them a little more

information about the type of information you'd like them to provide.
Let's take a look at the placeholder attribute next.

Create New Account

First Name
John

Last Name
Smith

Email
user@example.com

Password
8-10 characters

Password Confirmation
Type your password again

Sign Up

Figure 3.3: Placeholders can help users understand what you're asking them to do

6

Providing Hints with Placeholder Text

Placeholder text provides users with instructions on how they should fill in the fields. Figure 3.3 shows a signup form with placeholder text. We're going to construct that form now.

A simple signup form

AwesomeCo's support site requires users to sign up for an account, and one of the biggest problems with the signups is that users keep trying to use insecure passwords. Let's use placeholder text to give the users a little guidance on our password requirements. For consistency's sake, we'll add placeholder text to the other fields too.

To add placeholder text, you just add the placeholder attribute to each input field, like this:

[Download](#) html5placeholder.txt/index.html

```
<input id="email" type="email"
      name="email" placeholder="user@example.com">
```

Our entire form's markup looks something like this, with placeholder text for each field.

[Download](#) html5placeholder.txt/index.html

```
<form id="create_account" action="/signup" method="post">
  <fieldset id="signup">
    <legend>Create New Account</legend>
    <ol>
      <li>
        <label for="first_name">First Name</label>
        <input id="first_name" type="text"
              autofocus="true"
              name="first_name" placeholder="John">
      </li>
      <li>
        <label for="last_name">Last Name</label>
        <input id="last_name" type="text"
              name="last_name" placeholder="Smith">
      </li>
      <li>
        <label for="email">Email</label>
        <input id="email" type="email"
              name="email" placeholder="user@example.com">
      </li>
      <li>
        <label for="password">Password</label>
        <input id="password" type="password" name="password" value=""
              autocomplete="off" placeholder="8-10 characters" />
      </li>
      <li>
        <label for="password_confirmation">Password Confirmation</label>
        <input id="password_confirmation" type="password"
              name="password_confirmation" value=""
              autocomplete="off" placeholder="Type your password again" />
      </li>
    </ol>
  </fieldset>
</form>
```



```

        <li><input type="submit" value="Sign Up"></li>
    </ol>
</fieldset>
</form>

```

The autocomplete attribute

You may have noticed we've added the autocomplete attribute to the password fields on this form. HTML5 introduces an autocomplete attribute that tells web browsers that they should not attempt to automatically fill in data for the field. Some browsers remember data that users have previously typed in, and in some cases, we want to tell the browsers that we'd rather not let users do that.

Since we're once again using the ordered list element to hold our form fields, we'll add a bit of basic CSS to make the form look nicer.

[Download](#) `html5placeholderext/style.css`

```

fieldset{
    width: 216px;
}

fieldset ol{
    list-style: none;
    padding:0;
    margin:2px;
}

fieldset ol li{
    margin:0 0 9px 0;
    padding:0;
}

/* Make inputs go to their own line */
fieldset input{
    display:block;
}

```

Now, users of Safari, Opera, and Chrome will have helpful text inside of the form fields. Now let's make Firefox and Internet Explorer play along.

Falling Back

You can use JavaScript to put placeholder text on form fields without too much work. You test the value of each form field and if it's empty, you set its value to the placeholder value. When the form receives focus, you clear out the value, and when the field loses focus, you test the

value again. If it's different, you leave it alone, and if it's empty, you replace it with the placeholder text.

You test for placeholder support just like you test for autofocus support.

Download <html5placeholdertext/index.html>

```
function hasPlaceholderSupport() {
  var i = document.createElement('input');
  return 'placeholder' in i;
}
```

Then you just write your JavaScript to handle the changes. We'll use a solution based on work by Andrew January⁴ and others to make this work. We'll fill in the values of all form fields with the text stored in the placeholder attribute. When a user selects a field, we'll remove the text we placed in the field. Let's wrap this up in a jQuery plugin so that it's easy to apply the behavior to our form. See the sidebar on page 61 to learn how plugins work.

Download <html5placeholdertext/jquery.placeholder.js>

```
Line 1 (function($){
-
-   $.fn.placeholder = function(){
-
-     function valueIsPlaceholder(input){
5       return ($(input).val() == $(input).attr("placeholder"));
-     }
-     return this.each(function() {
-
10      $(this).find(":input").each(function(){
-
-        if($(this).attr("type") == "password"){
-
-          var new_field = $("<input type='text'>");
15          new_field.attr("rel", $(this).attr("id"));
-          new_field.attr("value", $(this).attr("placeholder"));
-          $(this).parent().append(new_field);
-          new_field.hide();
-
20          function showPasswordPlaceholder(input){
-            if( $(input).val() == "" || valueIsPlaceholder(input) ){
-              $(input).hide();
-              $('input[rel=' + $(input).attr("id") + ']').show();
-            };
25          };
-        }
-      });
-    }
-  }
- }
- }
- }
```

4. The original script is at <http://www.morethanotherthing.co.uk/wp-content/uploads/2010/01/placeholder.js> but didn't support password fields in IE.

```

-
-     new_field.focus(function(){
-         $(this).hide();
-         $('input#' + $(this).attr("rel")).show().focus();
30     });
-
-     $(this).blur(function(){
-         showPasswordPlaceholder(this, false);
-     });
35
-     showPasswordPlaceholder(this);
-
- }else{
-
40     // Replace the value with the placeholder text.
-     // optional reload parameter solves FF and IE caching values on fields.
-     function showPlaceholder(input, reload){
-         if( $(input).val() == "" || ( reload && valueIsPlaceholder(input) ) ){
-             $(input).val($(input).attr("placeholder"));
45         }
-     };
-
-     $(this).focus(function(){
-         if($(this).val() == $(this).attr("placeholder")){
50             $(this).val("");
-         };
-     });
-
-     $(this).blur(function(){
55         showPlaceholder($(this), false)
-     });
-
-     showPlaceholder(this, true);
60 };
- });
-
- // Prevent forms from submitting default values
- $(this).submit(function(){
65     $(this).find(":input").each(function(){
-         if($(this).val() == $(this).attr("placeholder")){
-             $(this).val("");
-         }
-     });
70 });
-
- });
-
75 })(jQuery);

```

There are a couple of interesting things in this plugin that you should know about. On line 43, we're reloading the placeholder text into the fields if they have no value, but also if we've refreshed the page. Firefox and other browsers persist the values of forms. We're setting the value attribute to the placeholder, and we certainly don't want that to accidentally become the user's actual value. When we load the page, we pass true to this method, which you can see on line 59.

Password fields behave a little differently than other form fields, so we have to handle those differently as well. Take a look at line 12. We're detecting the presence of a password field, and we have to change its type to a regular text field so that the value doesn't show up masked with asterisks. Some browsers throw errors if you try to convert password fields, so we'll have to swap out the password field for a text field. We'll swap those fields in and out as the user interacts with the fields.

This hack changes the values on the forms, and you probably want to prevent those placeholders from making their way back to the server. Since we're only hacking in this placeholder code when JavaScript is enabled, we can use JavaScript to inspect the form submission and strip out any values that match the placeholder text. On line 64, we capture the form submission and clear out the values of any input fields that equal the placeholder values.

Now that it's all written up as a plugin, we can invoke it on the page by attaching it to the form like this:

[Download](#) `html5placeholdertext/index.html`

```
$(function(){
  function hasPlaceholderSupport() {
    var i = document.createElement('input');
    return 'placeholder' in i;
  }

  if(!hasPlaceholderSupport()){
    $("#create_account").placeholder();
    //END placeholder_fallback

    $('input[autofocus=true]').focus();
  }
});
```

Now we've got a pretty decent solution that makes placeholder text a viable option for your web apps, no matter what browser you use.

jQuery Plugins

You can extend jQuery by writing your own plugins. You add your own methods on to the jQuery function, and your plugin seamlessly becomes available to any developer that includes your library. Here's a really trivial example that displays a JavaScript alert box:

```
jQuery.fn.debug = function() {
  return this.each(function(){
    alert(this.html());
  });
};
```

If you wanted to see a popup box appear for every paragraph on the page, you'd call it like this:

```
$("#p").debug();
```

jQuery plugins are designed to iterate over a collection of jQuery objects, and they also return that object collection so that you can chain them. For example, since our debug plugin also returns the jQuery collection, we can use jQuery's `css` method to change the color of the text of these paragraphs, all on one line.

```
$("#p").debug().css("color", "red");
```

We'll make use of jQuery plugins a few times throughout this book to help us keep our code organized when we create fallback solutions. You can learn more at jQuery's documentation site.*

*, <http://docs.jquery.com/Plugins/Authoring>

7 In-Place Editing with ContentEditable

We're always looking for ways to make it easier for people to interact with our applications. Sometimes we want a user of our site to edit some information about themselves without having to navigate to a different form. In-place editing traditionally involves replacing a text area with an input field using JavaScript and event listeners. HTML5 has built-in support for in-place editing of text. We'll still have to write

User information

Name	Hugh Mann
City	Anytown
State	OH
Postal Code	92110
Email	boss@awesomecompany.com

Figure 3.4: In Place Editing Made Easy

some JavaScript to send the data back to the server so we can save it, but we no longer have to create and toggle hidden forms.

One of your current projects lets users review their account profile. It displays their name, city, state, postal code, and email address. Let's add some in-place editing to this profile page, so that we end up with an interface like Figure 3.4.

I wanted to get to the meat of this tip early so that you could see how incredibly useful the contenteditable chapter could be. However, it goes against everything I believe in when it comes to building accessible web applications. Always, and I mean *always* build the solution that does not require JavaScript, and *then* build the version that relies on scripting.

And be sure to write automated tests for both paths so that you're more likely to catch bugs if you change one version and not the other.

The Profile Form

HTML5 introduces the contenteditable attribute which is available on almost every element. Simply adding this attribute turns it into an editable field.

[Download](#) `html5_content_editable/show.html`

```
<h1>User information</h1>
<div id="status"></div>
<ul>
  <li>
    <b>Name</b>
    <span id="name" contenteditable="true">Hugh Mann</span>
  </li>
  <li>
```

```

    <b>City</b>
    <span id="city" contenteditable="true">Anytown</span>
  </li>
  <li>
    <b>State</b>
    <span id="state" contenteditable="true">OH</span>
  </li>
  <li>
    <b>Postal Code</b>
    <span id="postal_code" contenteditable="true">92110</span>
  </li>
  <li>
    <b>Email</b>
    <span id="email" contenteditable="true">boss@awesomecompany.com</span>
  </li>
</ul>

```

We can style this up with some CSS too. We'll use some CSS3 selectors to identify the editable fields so they change color when our users hover over or select them.

[Download](#) html5_content_editable/show.html

```

Line 1 ul{list-style:none;}
-
- li{clear:both;}
-
5 li>b, li>span{
-   display: block;
-   float: left;
-   width: 100px;
- }
10
- li>span{
-   width:500px;
-   margin-left: 20px;
- }
15
- li>span[contenteditable=true]:hover{
-   background-color: #ffc;
- }
-
20 li>span[contenteditable=true]:focus{
-   background-color: #ffa;
-   border: 1px shaded #000;
- }

```

Persisting the data

While the users can change the data, their changes will be lost if they refresh the page or navigate away. We need a way to submit those

changes to our backend, and we can do that easily with some JQuery. If you've ever done any AJAX before, this won't be anything new to you.

[Download](#) `html5_content_editable/show.html`

```
$(function(){
    var status = $("#status");
    $("span[contenteditable=true]").blur(function(){
        var field = $(this).attr("id");
        var value = $(this).text();
        $.post("http://localhost:4567/users/1",
            field + "=" + value,
            function(data){
                status.text(data);
            }
        );
    });
});
```

We'll add an event listener to every span on the page that has the `contenteditable` attribute set to `true`. Then, all we have to do is submit the data to our server-side script.

Falling Back

We've done a bunch of things that won't work for some of our audience. First, we've created a dependency on JavaScript to save the edited results back to the server, which is a Bad Thing. Next, we're using the Focus pseudo class to highlight the fields when they receive focus, and some versions of IE don't support that. Let's handle the functionality first and then we'll deal with the visual effects.

Creating an Edit page

Rather than worrying too much about various situations that might prevent a user from using our technique, let's just give them the option to go to a separate page with its own form. Sure, it's more coding, but think about the possible scenarios:

1. A user doesn't have JavaScript turned on and is using Internet Explorer 7
2. A user doesn't have an HTML5 compatible browser
3. A user is using the latest Firefox with HTML5 support but still disabled JavaScript simply because they don't like JavaScript (it happens all the time. More than you'd think.)

When it comes down to it, making a form that does a POST to the same action that handled the AJAX update makes the most sense. How you do this is up to you, but many frameworks let you detect the type of request by looking at the accept headers to determine whether the request came from a regular POST or an XMLHttpRequest. That way you keep the server-side code DRY⁵. We will hide the link to this form if the browser supports Contenteditable and JavaScript.

So, create a new page called `edit.html` and code up a standard edit form that posts to the same update action that our AJAX version uses.

[Download](#) `html5_content_editable/edit.html`

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Editing Profile</title>
  </head>
  <body>
    <form action="/users/1" method="post" accept-charset="utf-8">
      <fieldset id="your_information">
        <legend>Your Information</legend>
        <ol>
          <li>
            <label for="name">Your Name</label>
            <input type="text" name="name" value="" id="name">
          </li>
          <li>
            <label for="city">City</label>
            <input type="text" name="city" value="" id="city">
          </li>
          <li>
            <label for="state">State</label>
            <input type="text" name="state" value="" id="state">
          </li>
          <li>
            <label for="postal_code">Postal Code</label>
            <input type="text" name="postal_code" value="" id="postal_code">
          </li>
          <li>
            <label for="email">Email</label>
            <input type="email" name="email" value="" id="email">
          </li>
        </ol>
      </fieldset>
    </form>
  </body>
</html>
```

5. DRY stands for "Don't Repeat Yourself", and is a term coined by Dave Thomas and Andy Hunt in *The Pragmatic Programmer* [HT00]

```

    <p><input type="submit" value="Save"></p>
  </form>

  </body>
</html>

```

Then, add a link to this page on show.html.

[Download](#) html5_content_editable/show.html

```

<h1>User information</h1>
<section id="edit_profile_link">
  <p><a href="edit.html">Edit Your Profile</a></p>
</section>
<div id="status"></div>

```

With the link added, we just need to modify our script a bit. We want to hide the link to the edit page and enable the AJAX support only if we have support for editable content.

[Download](#) html5_content_editable/show.html

```
if(document.getElementById("edit_profile_link").contentEditable != null){
```

With the detection in place, our script looks like this:

[Download](#) html5_content_editable/show.html

```

$(function(){
  if(document.getElementById("edit_profile_link").contentEditable != null){
    $("#edit_profile_link").hide();
    var status = $("#status");
    $("span[contenteditable=true]").blur(function(){
      var field = $(this).attr("id");
      var value = $(this).text();
      $.post("http://localhost:4567/users/1",
        field + "=" + value,
        function(data){
          status.text(data);
        }
      );
    });
  }
});

```

With that in place, our users have the ability to use a standard interface or a quicker "in-place" mode. Now that you know how to implement this interface, remember to implement the fallback solution first. Unlike the other fallback solutions, this particular one cripples functionality if not implemented.

The Future

Right now, if you add a JavaScript-based date picker to your site, your users have to learn how it works. If you've ever shopped online for plane tickets and made hotel reservations, you're already familiar with the different ways people implement custom form controls on sites. It's akin to using an ATM—the interface is often different enough to slow you down.

Imagine though if each web site used the HTML5 date field, and the browser had to create the interface. Each site a user visited would display the exact same date picker. Screen reading software could even implement a standard mechanism to allow the blind to enter dates easily. Now think about how useful placeholder text and autofocus can be for users once it's everywhere. Placeholder text can help screenreaders explain to users how form fields should work, and autofocus could help people navigate more easily without a mouse, which is handy for the blind, but also for users with motor impairments that may not use the mouse.

The ability for developers to turn any element into an editable region makes it easy to do in-place editing, but it could potentially change how we build interfaces for content management systems.

The modern Web is all about interactivity, and forms are an essential part of that interactivity. The enhancements provided by HTML5 give us a whole new set of tools we can use to help our users.

Making Better User Interfaces with CSS3

For far too long, we developers have hacked around CSS to get the effects we need in our code. We've used JavaScript or server-side code to stripe table rows or put focus and blur effects on our forms. We've had to litter our tags with additional class attributes just so we could identify which of our fifty form inputs we want to style.

But no more! CSS3 has some amazing selectors that make some of this work trivial. In case you forgot, a selector is a pattern that you use to help you find elements in the HTML document so you can apply styles to those elements. We'll use these new selectors to style a table. Then we'll take a look at how we can use some other CSS3 features to improve our site's print stylesheets, and we'll split content into multiple columns.

Feature	Description	Use	Supported Browsers
:nth-of-type	Finds all n elements of a certain type	p:nth-of-type(2n+1){color:red;}	<ul style="list-style-type: none"> • Safari 3+ • Firefox 3.5+ • Chrome 2+ • Opera 9.5+
:first-child	Finds the first child element	p:first-child{color:blue;}	<ul style="list-style-type: none"> • Safari 3+ • Firefox 3.5+ • Chrome 2+ • Opera 9.5+
:nth-child	Finds a specific child element counting forward	p:nth-child(2n+1){color:red;}	<ul style="list-style-type: none"> • Safari 3+ • Firefox 3.5+ • Chrome 2+ • Opera 9.5+
:last-child	Finds the last child element	p:last-child{color:blue;}	<ul style="list-style-type: none"> • Safari 3+ • Firefox 3.5+ • Chrome 2+ • Opera 9.5+
:nth-last-child	Finds a specific child element counting backward	p:nth-last-child(2){color:red;}	<ul style="list-style-type: none"> • Safari 3+ • Firefox 3.5+ • Chrome 2+

8

Styling Tables With Pseudo Classes

A “pseudo class” in CSS is a way to select elements based on information that lies outside of the document, or information that can’t be expressed using normal selectors. You’ve probably used pseudo classes like `:hover` before to change the color of a link when the user hovers over it with his or her mouse pointer. CSS3 has several new pseudo classes that make locating elements much easier.

Improving an invoice

AwesomeCo uses a third-party billing and invoicing system for products they ship. You see, one of AwesomeCo’s biggest markets is conference swag, like pens, cups, shirts, and anything else you can slap your logo on. You’ve been asked to make the invoice more readable. Right now, the developers are producing a standard HTML table that looks like the one in Figure 4.1, on the next page.

It’s a pretty standard invoice with prices, quantities, row totals, a subtotal, shipping total, and a grand total for the order. It would be easier to read if every other row was colored differently. It would also be helpful if the grand total was a different color so that it stands out more.

The code for the table looks like this. Copy it into your own file so you can work with it.

[Download](#) `css3advancedselectors/table.html`

```
<table >
  <tr>
    <th>Item</th>
    <th>Price</th>
    <th>Quantity</th>
    <th>Total</th>
  </tr>
  <tr>
    <td>Coffee mug</td>
    <td>$10.00</td>
    <td>5</td>
    <td>$50.00</td>
  </tr>
  <tr>
    <td>Polo shirt</td>
    <td>$20.00</td>
```

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
Subtotal			\$186.00
Shipping			\$12.00
Total Due			\$198.00

Figure 4.1: The current invoice uses an unstyled HTML table.

```

        <td>5</td>
        <td>$100.00</td>
    </tr>
    <tr>
        <td>Red stapler</td>
        <td>$9.00</td>
        <td>4</td>
        <td>$36.00</td>
    </tr>
    <tr>
        <td colspan="3">Subtotal</td>
        <td>$186.00</td>
    </tr>
    <tr>
        <td colspan="3">Shipping</td>
        <td>$12.00</td>
    </tr>
    <tr>
        <td colspan="3">Total Due</td>
        <td>$198.00</td>
    </tr>

```

</table>

First, let's get rid of the hideous default table border.

Download [css3advancedselectors/table.css](#)

```
table{
  width: 600px;
  border-collapse: collapse;
}

th, td{
  border: none;
}
```

We'll also style the header a bit by giving it a black background with white text.

Download [css3advancedselectors/table.css](#)

```
table th{
  background-color: #000;
  color: #fff;
}
```

Apply that style, and the table looks like this:

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
Subtotal			\$186.00
Shipping			\$12.00
Total Due			\$198.00

With the table's borders and spacing cleaned up a bit, we can start using the pseudo classes to style individual rows and columns. We'll start by striping the table.

Striping rows with :nth-of-type

Adding "zebra striping" to tables is something we've all seen. It's useful because it gives users horizontal lines to follow. This kind of styling is best done in CSS, the presentation layer, but we don't want to pollute the markup for this table with class names like "odd" and "even", as the HTML5 specification encourages us to avoid using class names that define presentation.

The *nth-of-type* selector finds every *nth* element of a specific type using either a formula or keywords. We'll get into the formula in more detail soon, but first, let's focus on the keywords, as they're immediately easier to grasp.

We want to stripe every other row of the table with a different color, and the easiest way to do that is to find every even row of the table and give it a background color. We then do the same thing with the odd rows. CSS3 has even and odd keywords that support this exact situation.

[Download](#) `css3advancedselectors/table.css`

```
table tr:nth-of-type(even){
  background-color: #F3F3F3;
}
table tr:nth-of-type(odd) {
  background-color:#ddd;
}
```

So this selector says “Find me every even table row and color it. Then find every odd row and color that too.”. That takes care of our zebra striping, without resorting to any scripting or extra class names on rows.

With the styles applied, our table looks like this:

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
Subtotal			\$186.00
Shipping			\$12.00
Total Due			\$198.00

Now let's work on aligning the columns in the table.

Aligning Column Text with `:nth-child`

By default, all of the columns in our invoice table are left-aligned. Let's right align every column except for the first column. This way, our price and quantity columns will be right-aligned and easier to read. To do that, we can use `nth-child`, but first we have to learn how it works.

The `nth-child` selector looks for child elements of an element and, like `nth-of-type` can use keywords or a formula.

The formula is a_n+b , where b is the offset, and a is a multiple. That description is not particularly helpful without some context, so let's look at it in the context of our table.

If we wanted to select all of the table rows, we could use this selector:

```
table tr:nth-child(n)
```

We're not using any multiple, nor are we using an offset.

However, if we wanted to select all rows of the table except for the first row, which is the row containing the column headings, we would use this selector that uses an offset:

```
table tr:nth-child(n+2)
```

The counter is 1-based, not zero-based as you might expect if you're used to working with arrays in JavaScript or other languages. We need to find all rows, starting with the second row.

If we wanted to select every other row of our table, we'd use a multiple, or $2n$.

```
table tr:nth-child(2n)
```

If you wanted every third row, you'd use $3n$.

You can also use the offset, so that you can start further down the table. This selector would find every other row, starting with the fourth row:

```
table tr:nth-child(2n+4)
```

So, we can align every column *except* the first one with this rule:

[Download](#) `css3advancedselectors/table.css`

```
table td:nth-child(n+2){
  text-align: right;
}
```

At this point, our table is really shaping up:

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
Subtotal			\$186.00
Shipping			\$12.00
Total Due			\$198.00

Now, let's style the last row of the table.

Bolding the Last Row With `:last-child`

The invoice is looking pretty good right now, but one of the managers would like the bottom row of the table to be bolder than the other rows so it stands out more. We can use `last-child` for that too, which grabs the last child in a group.

Applying a bottom margin to paragraphs so that they are evenly spaced on a page is a common practice among many web developers. This can sometimes lead to an extra bottom margin at the end of a group, and that might be undesirable. For example, if the paragraphs are sitting inside of a sidebar or callout box, we may want to remove the bottom margin from the last paragraph so that there's not wasted space between the bottom of the last paragraph and the border of the box. The `last-child` selector is the perfect tool for this. We can use it to remove the margin from the last paragraph.

```
p{margin-bottom: 20px}
#sidebar p:last-child{ margin: 0; }
```

Let's use this same technique to bold the contents of the last row.

[Download](#) `css3advancedselectors/table.css`

```
table tr:last-child{
  font-weight: bolder;
}
```

Let's do the same thing with the last column of the table. This will help the line totals stand out too.

[Download](#) `css3advancedselectors/table.css`

```
table td:last-child{
  font-weight: bolder;
}
```

Finally, we'll make the total's font size bigger by using `last-child` with descendant selectors. We'll find the last column of the last row and style it with this:

[Download](#) `css3advancedselectors/table.css`

```
table tr:last-child td:last-child{
  font-size:24px;
}
```

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
Subtotal			\$186.00
Shipping			\$12.00
Total Due			\$198.00

We're almost done, but there are a few things left to do with the last three rows of the table.

Counting Backwards with `:nth-last-child`

We'd like to highlight the shipping row of the table when there's a discounted shipping rate. We'll use `nth-last-child` to quickly locate that row. You saw how you can use `nth-child` and the formula `an+b` to select specific child elements in Section 8, *Aligning Column Text with `:nth-child`*, on page 73. The `nth-last-child` selector works exactly the same way, except that it counts backwards through the children, starting at the last child first. This makes it easy to grab the the second to the last element in a group. It turns out that we need to do just that with our invoice table.

So, to find our shipping row, we'd use this code:

[Download](#) `css3advancedselectors/table.css`

```
table tr:nth-last-child(2){
  color: green;
}
```

Here, we're just specifying a specific child, the second to the last.

There's one last thing we should do with this table though. Earlier, we right-aligned all of the columns except for the first column, and while that looks fine for the rows of the table with the item descriptions and prices, it makes the last three rows of the table look a little funny. Let's right-align the bottom three rows as well. We can do that by using `nth-`

Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
		Subtotal	\$186.00
		Shipping	\$12.00
		Total Due	\$198.00

Figure 4.2: Our styled table, with striping and alignment done entirely with CSS3.

last-child with a negative value for n and a positive value for α in our formula, like this:

```
Download css3advancedselectors/table.css
tr:nth-last-child(-n+3) td{
  text-align: right
}
```

You can think of this as a range selector... it's using the offset of 3, and since we're using `nth-last-child`, it's grabbing every element before the offset. If you were using `nth-child`, this formula would grab every row up to the offset.

Our newly-styled table, shown in Figure 4.2, looks much better now, and we didn't have to change the underlying markup one bit. Many of the selectors we used to accomplish this are not yet available to people using Internet Explorer, so we need a workaround for them.

Falling Back

Current versions of Opera, Firefox, Safari, and Chrome all understand these selectors, but Internet Explorer versions 8.0 and lower will just ignore these entirely. You'll need a good fallback solution, and you have a choice to make.

Change the HTML code

The most obvious solution that works everywhere is to modify the underlying code. You could attach classes to all of the cells in the table and apply basic CSS to each class. This is the worst choice, because it mixes presentation and content, and is exactly the kind of thing we're using

CSS3 to avoid. Someday we won't need all that extra markup, and it would be painful to remove it.

Use JavaScript

The jQuery library already understands most of the CSS3 selectors we used, so we could quickly write a method to style the table that way, but there's an easier way.

Keith Clark has written a great little library called IE-css3¹ that adds support for CSS3 selectors to Internet Explorer. All we need to do is add a couple of scripts to our page.

The IE-CSS3 library can use jQuery, Prototype, or several other libraries under the hood, but I prefer to use DOMAssistant² library because it has the best support for all of the pseudo-classes we've used here.

Download both of those libraries and then link them to your document. Since this is for IE only, you can place them in a conditional comment so they'll only be used by your IE users.

[Download](#) `css3advancedselectors/table.html`

```
<!--[if (gte IE 5.5)&(!te IE 8)]>
  <script type="text/javascript" src="js/DOMAssistantCompressed-2.8.js"></script>
  <script type="text/javascript" src="js/ie-css3.js"></script>
<![endif]-->
```

Placing those scripts in the page makes things look just great in Internet Explorer. You can see what it looks like in Figure 4.3, on the following page .

While this will require the user to have JavaScript turned on, the table styling is mainly there to make the content easier to see. Lack of styling doesn't prevent anyone from reading the invoice.

Styling elements is a whole lot easier with CSS3, especially if we don't have the ability to modify the HTML we're targeting. When you're styling interfaces, use the semantic hierarchy and these new selectors before you add additional markup. You'll find your code much easier to maintain.

1. <http://www.keithclark.co.uk/labs/ie-css3/>
 2. <http://www.domassistant.com/>



Item	Price	Quantity	Total
Coffee mug	\$10.00	5	\$50.00
Polo shirt	\$20.00	5	\$100.00
Red stapler	\$9.00	4	\$36.00
		Subtotal	\$186.00
		Shipping	\$12.00
		Total Due	\$198.00

Figure 4.3: Our table looks great in Internet Explorer

9

Making Links Printable with :after and content

CSS can style existing elements, but it can also inject content into a document. There are a few cases where content generation with CSS makes sense, and the most obvious one is appending the URL of a hyperlink next to the link's text when a user prints the page. When you're looking at a document on the screen, you can just hover over a link and see where it goes by looking at the status bar. However, when you look at a printout of a page, you have absolutely no idea where those links go.

AwesomeCo is working up a new page for its forms and policies, and one of the members of the redesign committee insists on printing out a copy of the site each time. He wants to be able to know exactly where all of the links go on the page so that he can determine if they need to be moved. With just a little bit of CSS, we can add that functionality, and it will work in IE 8, Firefox, Safari, and Chrome. We can use some proprietary JavaScript to make it work in IE 6 and 7.

The page itself has nothing more than a list of links on it right now. Eventually it'll get put into a template.

Download css3_print_links/index.html

```
<ul>
  <li>
```

```

    <a href="travel/index.html">Travel Authorization Form</a>
  </li>
  <li>
    <a href="travel/expenses.html">Travel Reimbursement Form</a>
  </li>
  <li>
    <a href="travel/guidelines.html">Travel Guidelines</a>
  </li>
</ul>

```

```
</body>
```

If you were to look at that page on a printout, you'd have no idea where those links go. Let's fix that.

The CSS

When we add a stylesheet to a page, we can specify the media type that the styles apply to. Most of the time, we use the screen type. However, we can use the print type to define a stylesheet that only loads when the page is printed (or when the user uses the print preview function).

[Download](#) `css3_print_links/index.html`

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

We can then create a print.css stylesheet file with this simple rule:

[Download](#) `css3_print_links/print.css`

```

a:after {
  content: " (" attr(href) ") ";
}

```

This takes every link on the page and adds the value of the href value inside of parentheses after the link's text. When you open it in a modern browser, it looks just like this:

Forms and Policies

- [Travel Authorization Form \(travel/index.html\)](#)
- [Travel Reimbursement Form \(travel/expenses.html\)](#)
- [Travel Guidelines \(travel/guidelines.html\)](#)

That handles everything except for Internet Explorer 6 and 7. Let's fix that, shall we?

Falling Back

Internet Explorer has a couple of JavaScript events that I wish every browser would adopt: `onbeforeprint` and `onafterprint`. Using those events, we can modify the hyperlink text when the printing is triggered and then revert the links back when printing is finished. Our users will never notice the difference.³

We just need to create a file called `print.js` and add this code:

Download `css3_print_links/print.js`

```
Line 1 $(function() {
-     if (window.onbeforeprint !== undefined) {
-         window.onbeforeprint = ShowLinks;
-         window.onafterprint = HideLinks;
5     }
- });
-
-     function ShowLinks() {
-         $("a").each(function() {
10             $(this).data("linkText", $(this).text());
-             $(this).append(" (" + $(this).attr("href") + ")");
-         });
-     }
-
15     function HideLinks() {
-         $("a").each(function() {
-             $(this).text($(this).data("linkText"));
-         });
-     }
}
```

Then we just need to attach it to our page. We only need this fallback for IE 6 and 7, so we'll use a conditional comment for that. This code relies on jQuery, so we have to make sure that we link in the jQuery library as well.

Download `css3_print_links/index.html`

```
<script
  charset="utf-8"
  src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js'
  type='text/javascript'>
</script>
<!--[if lte IE 7]>
  <script type="text/javascript" src="print.js"></script>
<![endif]-->
</head>
<body>
```

3. This technique is outlined nicely at <http://beckelman.net/post/2009/02/16/Use-jQuery-to-Show-a-Link-Address-After-its-Text>

```
<h1>Forms and Policies</h1>

<ul>
  <li>
    <a href="travel/index.html">Travel Authorization Form</a>
  </li>
  <li>
    <a href="travel/expenses.html">Travel Reimbursement Form</a>
  </li>
  <li>
    <a href="travel/guidelines.html">Travel Guidelines</a>
  </li>
</ul>
```

With the JavaScript linked, the link URLs will print on all of our target browsers. You can use this print stylesheet as the basis for a more comprehensive one, and you may choose to only apply this behavior to some links on your site, and not to every link like we did here.

Creating Multi-Column Layouts

The print industry has had columns for years, and web designers have looked at those publications with envy. Narrow columns make it easier for readers to read your content, and with displays getting wider, developers are looking for ways to preserve comfortable column widths. After all, nobody wants to follow multiple lines of text across the monitor any more than they want a line of text to flow across the whole page of a newspaper. There have been some pretty clever solutions in the past ten years, but none of those are as simple and easy as the method provided by the CSS3 specification.

Splitting Columns

Each month, AwesomeCo publishes a newsletter for its employees. The company happens to use a popular web-based email system. Email-based newsletters don't quite look good and are very hard to maintain. They've decided to put the newsletter on the Intranet site and are planning to just send emails to employees with a link to pull up the newsletter in their browsers. See Figure 4.4, on the next page for a mocked-up version of this new newsletter.

The new director of communications, who has a background in print publications, has decided that she would like the newsletter to look more like an actual newsletter, with two columns instead of one.

If you've ever tried to split some text into multiple columns using divs and floats, you know how hard that can be. The first big hurdle you run into is that you have to manually decide where to split the text. In publishing software like InDesign, you can "link" text boxes together so that when one fills up with text, the text flows into the linked text area. We don't have anything quite like that on the Web just yet, but we have something that works really well, and is quite easy to use. We can take an element and split its contents into multiple columns, each with the same width.

We'll start with the markup for the newsletter. It's fairly basic HTML. Since its content will change once it's written, we're just going to use placeholder text for the content. If you're wondering why we're not using the new HTML5 markup elements like `section` and `such` for this newslet-

AwesomeCo Newsletter

Volume 3, Issue 12

News From The Director

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Quick Bits of Awesome

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Birthdays

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Send newsworthy things to news@awesomeco.com.

Being Awesome

"Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam."

Figure 4.4: Our single-column newsletter is harder to read as it's very wide.

ter, it's because our fallback method isn't compatible with those elements in Internet Explorer.

[Download](#) `css3columns/condensed_newsletter.html`

```
<body>
  <div id="header">
    <h1>AwesomeCo Newsletter</h1>
    <p>Volume 3, Issue 12</p>
  </div>
  <div id="newsletter">
    <div id="director_news">
      <div>
        <h2>News From The Director</h2>
      </div>
      <div>
        <p>
          Lorem ipsum dolor...
        </p>
        <div class="callout">
          <h4>Being Awesome</h4>
          <p>
            &quot;Lorem ipsum dolor sit amet...&quot;
          </p>
        </div>
      </div>
    </div>
  </div>
</body>
```

```

        </p>
    </div>
    <p>
        Duis aute irure...
    </p>
</div>
</div>

<div id="awesome_bits">
    <div>
        <h2>Quick Bits of Awesome</h2>
    </div>
    <div>
        <p>
            Lorem ipsum...
        </p>
        <p>
            Duis aute irure...
        </p>
    </div>
</div>

<div id="birthdays">
    <div>
        <h2>Birthdays</h2>
    </div>
    <div>
        <p>
            Lorem ipsum dolor...
        </p>
        <p>
            Duis aute irure...
        </p>
    </div>
</div>

</div>
<div id="footer">
    <h6>Send newsworthy things to
        <a href="mailto:news@aweseomco.com">news@awesomeco.com</a>.
    </h6>
</div>
</body>

```

To split this into a two-column layout, all we need to do is add this to our stylesheet:

[Download](#) css3columns/newsletter.html

```

#newsletter{
    -moz-column-count: 2;
    -webkit-column-count: 2;
}

```

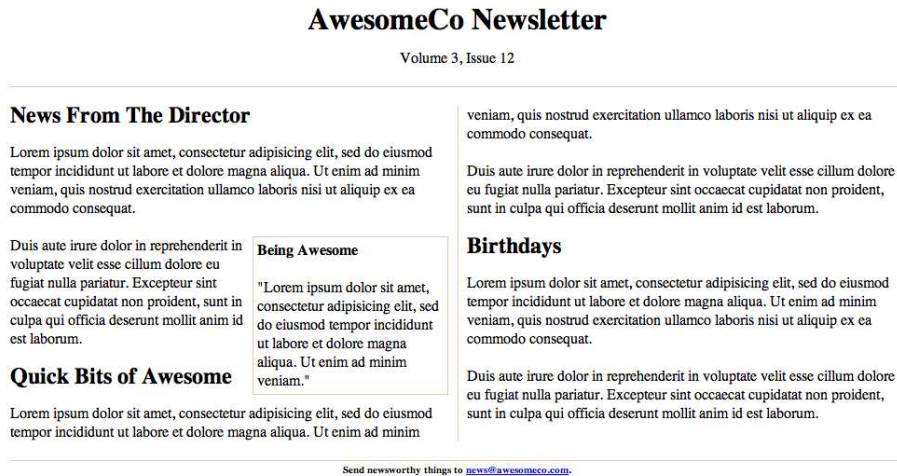


Figure 4.5: Our new two-column newsletter

```

-moz-column-gap: 20px;
-webkit-column-gap: 20px;
-moz-column-rule: 1px solid #ddccb5;
-webkit-column-rule: 1px solid #ddccb5;
}

```

Now we have something much nicer, like you see in Figure 4.5. We can add in more content and the browser will automatically determine how to split the content evenly. Also, notice that the floated elements float to the columns that contain them.

Falling Back

CSS3 columns don't work in Internet Explorer 8 and below, so we'll use the jQuery Columnizer plugin⁴ as a fallback. Columnizer will let us split our content evenly by simply using code like this:

[Download](#) `css3columns/newsletter.html`

```
$("#newsletter").columnize({ columns: 2 });
```

People without JavaScript are going to be stuck with a single column of text, but they'll still be able to read the content, because we marked

4. <http://welcome.totheinter.net/columnizer-jquery-plugin/>



Joe Asks...

Can I specify different widths for each column?

Sorry, Joe, but you can't. I was a little surprised too at first, so I double-checked the specification, and at the time of writing, there was no provision for specifying multiple column widths.

However, when you think about how columns are traditionally used, it makes sense. Columns are not meant to be a hack to easily make a sidebar for your web site any more than tables are. Columns are meant to make reading long areas of text easier, and equal width columns are perfect for that.

it up in a linear fashion, so we've got them covered. However, we can use JavaScript to detect browser support for certain elements. If we retrieve a CSS property that exists, we'll get an empty string. If we get a null value back, we don't have that property available.

[Download](#) `css3columns/newsletter.html`

```
<script
  charset="utf-8"
  src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js'
  type='text/javascript'>
</script>

<script src="javascripts/autocolumn.js" charset="utf-8"
  type='text/javascript'></script>

<script type="text/javascript" charset="utf-8">
  function hasColumnSupport(){
    var element = document.documentElement;
    var style = element.style;
    if (style){
      return typeof style.columnCount == "string" ||
        typeof style.MozColumnCount == "string" ||
        typeof style.WebkitColumnCount == "string" ||
        typeof style.KhtmlColumnCount == "string";
    }
    return null;
  }

  $(function(){
    if(!hasColumnSupport()){
      $("#newsletter").columnize({ columns: 2 });
    }
  });
</script>
```

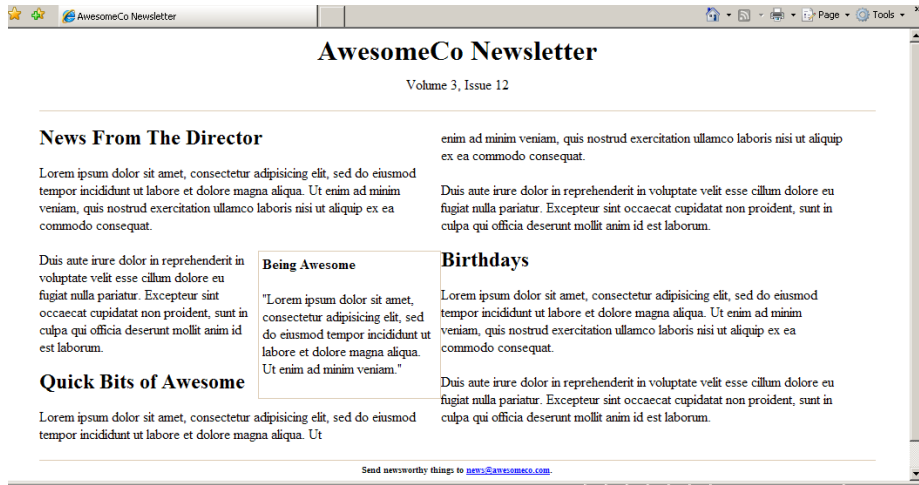


Figure 4.6: Our Internet Explorer version works, but needs some minor adjustments

```

    }
  });
</script>

```

We simply check for column support, and if none exists, we apply our plugin.

Refresh the page in Internet Explorer and you'll now see your two-column newsletter. It may not be perfect, as you can see in Figure 4.6, so you'll need to use a little CSS or JavaScript to fix any elements that don't quite look right, but I'm leaving that exercise up to you. Take advantage of conditional comments like we used in Section 8, *Use JavaScript*, on page 78 to target specific versions of Internet Explorer if needed.

Separating your content into multiple columns can make your content easier to read. However, if your page is longer, your users might find it annoying to have to scroll back to the top to read the columns. Use this with care.

The Future

The things we talked about in this chapter improve the user interface, but people can still work with our products if their browsers don't support these new features. People can still read the data in the table if it's not styled with stripes; the forms will still work, even if they don't have rounded corners on the interface elements; and the newsletter won't be laid out in multiple columns. It's good to know that we can use the presentation layer to achieve these effects instead of having to resort to JavaScript or server-side solutions.

Almost all browsers support these selectors now, with the exception of Internet Explorer. As we move forward, you can expect to see IE moving to support these as well, especially the pseudo classes. When the specification becomes final, the vendor-specific prefixes like `moz` and `webkit-` go away. Once that happens, you'll be able to remove your fallback code.

Improving Accessibility

Many of the new elements in HTML5 help you more accurately describe your content. This becomes more important when other programs start interpreting your code. For example, some people use software called screen readers to translate the graphical contents of the screen to text that's read aloud. Screen readers work by interpreting the text on the screen and the corresponding markup to identify links, images, and other elements. Screen readers have made amazing advances, but they're always lagging behind the current trends. Live regions on pages, where polling or AJAX requests alter content on the page, are difficult to detect. More complex pages can be difficult to navigate due to the screen reader needing to read a lot of the content aloud.

WIA-ARIA¹, or "Accessibility for Rich Internet Applications" is a specification that provides ways to improve the accessibility of web sites, especially web applications. It is especially useful if you're developing applications with JavaScript controls and AJAX. Some parts of the WIA-ARIA specification have been rolled into HTML5, while others remain separate and can complement the HTML5 specification. Many screen readers are already using features of the WIA-ARIA specification, including JAWS, WindowEyes, and even Apple's built-in VoiceOver feature. WIA-ARIA also introduces additional markup that assistive technology can use as hints for discovering regions that are updatable.

In this chapter, we'll see how HTML5 can improve the experience of your visitors who use these assistive devices. Most importantly, the techniques in this chapter require no fallback support, as many screen

1. <http://www.w3.org/WAI/intro/aria.php>

readers are already able to take advantage of these techniques right now.

Feature	Description	Use	Supported Browsers
role	Identifies responsibility of an element for screenreaders	<code><nav role="navigation"></code>	<ul style="list-style-type: none"> • IE 8 • Firefox 3.6 • Safari 4.0+ • Chrome 3.0+ • Opera 9.6
aria-live	Identifies a region that updates, possibly by AJAX	<code><div aria-live="polite"></code>	<ul style="list-style-type: none"> • IE 8 • Firefox 3.6 (Windows) • Safari 4.0+
aria-atomic	Identifies if the entire content of a live region should be read, or just the elements that changed.	<code><div aria-live="polite" aria-atomic="true"></code>	<ul style="list-style-type: none"> • IE 8 • Firefox 3.6 (Windows) • Safari 4.0+

11

Providing Navigation Hints with ARIA Roles

Most web sites share a common structure: there's a header, a navigation section, some main content, and a footer. Most of these sites are coded just like that, in a linear fashion. Unfortunately, this means that a screenreader may have to read the site to its user in that order. Since most sites repeat the same header and navigation on each page, the user will have to hear these elements each time they visit another page.

The recommended fix is to provide a hidden "skip navigation" link that screenreaders will read aloud, which simply links to an anchor somewhere near the main content. However, that's not something that's built in, and it's not something that everyone knows how (or remembers) to do.

HTML5's new "role" attribute lets us assign a "responsibility" to each element on your page. A screenreader can then very easily parse the page and categorize all of those responsibilities so that you could create a simple index for the page. For example, it can find all of the navigation roles on the page and present them to the user so he or she can quickly navigate around your application.

These roles come from the WIA-ARIA specification² and have been incorporated into the HTML5 specification. There are two specific classifications of roles that you can put to use right now: landmark roles and document roles.

Landmark Roles

Landmark roles identify "points of interest" on your site, such as the banner, search area, or navigation which screen readers can quickly identify.

2. <http://www.w3.org/WAI/PF/aria/roles>

Role	Use
banner	Identifies the banner area of your page.
search	Identifies the search area of your page
navigation	Identifies navigational elements on your page
main	Identifies where your page's main content begins
contentinfo	Identifies where information about the content exists, such as copyright information and publication date
complementary	Identifies content on a page that complements the main content, but is meaningful on its own.
application	Identifies a region of a page which contains a web application as opposed to a web document

We can apply a few of these roles to the AwesomeCo blog template we worked on in *Redefining a Blog using Semantic Markup*, on page 23.

For the overall header, let's apply the banner role like this:

[Download](#) html5_aria/blog.html

```
<header id="page_header" role="banner">
  <h1>AwesomeCo Blog!</h1>
</header>
```

All that's needed is the addition of the role="banner" to the existing header tag.

We can identify our navigation the same way:

[Download](#) html5_aria/blog.html

```
<nav role="navigation">
  <ul>
    <li><a href="/">Latest Posts</a></li>
    <li><a href="/archives">Archives</a></li>
    <li><a href="/contributors">Contributors</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
</nav>
```

The HTML5 specification says that some elements have default roles and can't be overridden. The nav element must have the role of navigation, and technically doesn't need to be specified. Screen readers aren't quite ready to accept that default yet, but many of them do understand these ARIA roles.

Our main and sidebar regions can be identified as follows:

[Download](#) html5_aria/blog.html

```
<section id="posts" role="main">
</section>
```

[Download](#) html5_aria/blog.html

```
<section id="sidebar"role="complementary">

  <nav>
    <h3>Archives</h3>
    <ul>
      <li><a href="2010/06">June 2010</a></li>
      <li><a href="2010/05">May 2010</a></li>
      <li><a href="2010/04">April 2010</a></li>
      <li><a href="2010/03">March 2010</a></li>
      <li><a href="2010/02">February 2010</a></li>
      <li><a href="2010/01">January 2010</a></li>
      <li><a href="2009/12">December 2009</a></li>
    </ul>
  </nav>
</section> <!-- sidebar -->
```

We identify the publication and copyright info in our footer using the contentinfo role like this:

[Download](#) html5_aria/blog.html

```
<footer id="page_footer" role="contentinfo">
  <p>&copy; 2010 AwesomeCo.</p>
</footer> <!-- footer -->
```

If we had a search for our blog, we could identify that region as well. Now that we've identified the landmarks, let's take this a step further and help identify some of the document elements.

Document Structure Roles

Document structure roles help screenreaders identify parts of static content easily, which can help better organize content for navigation.

Role	Use
document	Identifies a region containing document content, as opposed to application content.
article	Identifies a composition that forms an independent part of a document.
definition	Identifies a definition of a term or subject
directory	Identifies a list of references to a group, like a table of contents. Used for static content.
heading	Identifies a heading for a section of a page
img	Identifies a section that contains elements of an image. This may be image elements as well as captions and descriptive text.
list	Identifies a group of non-interactive list items/
listitem	Identifies a single member of a group of non-interactive list items.
math	Identifies a mathematical expression.
note	Identifies content that is parenthetical or ancillary to the main content of the resource.
presentation	Identifies content that is for presentation and can be ignored by assistive technology.
row	Identifies a row of cells in a grid.
rowheader	Identifies a cell containing header information for a row in a grid.

Many of the document roles are implicitly defined by HTML tags, like articles and headers. However, the document role isn't, and it's a very helpful role, especially in applications with a mix of dynamic and static content. For example, a web-based email client may have the document role attached to the element that contains the body of the email message. This is useful because screenreaders often have different methods for navigating using the keyboard. When the screenreader's focus is on an application element, it may need to allow keypresses through to the web application. However, when the focus is on static content, it could allow the screenreader's key bindings to work differently.

We can apply the document role to our blog by adding it to the body element.

[Download](#) html5_aria/blog.html

```
<body role="document">
```

This can help ensure that a screenreader will treat this page as static content.



Joe Asks...

Do we need these landmark roles if we have elements like nav and header?

The landmark roles may at first seem redundant but they provide you with the flexibility you need for situations where you can't use the new elements.

Using the search role, you can direct your users to the region of the page that not only contains the search field, but also links to a sitemap, a dropdown list of "quick links", or other elements that will help your users find information quickly, as opposed to just directing them to the actual search field.

There are also a lot more roles introduced by the specification than there are new elements and form controls.

Falling Back

These roles are already usable on the latest browsers with the latest screenreaders, so you can start working with them now. Browsers that don't support them are just going to ignore them, and so you're really only helping those people that can use them.

12

Creating An Accessible Updatable Region

We do a lot of things with AJAX in our web applications these days. Standard practice is to fire off some sort of visual effect to give the user a clue that something has changed on the page. However, a person using a screenreader obviously isn't going to be able to see any visual cues. The WIA-ARIA specification provides a pretty nice alternative solution which currently works in IE, Firefox, and Safari with various popular screen readers.

The AwesomeCo Executive Director of Communications wants a new home page. It should have links to a *services* section, a *contact* section, and an *about* section. He insists that the home page shouldn't scroll because "people hate scrolling." He would like you to implement a prototype for the page with a horizontal menu that changes the page's main content when clicked. That's easy enough to implement, and with the `aria-live` attribute, we can do something we haven't been able to do well before - implement this type of interface in a way that's friendly to screen readers.

Let's build a simple interface like Figure 5.1, on the following page. We'll put all of the content on the home page and if JavaScript is available to us, we'll hide all but the first entry. We'll make the navigation links point to each section using page anchors, and we'll use jQuery to change those anchor links into events that swap out the main content. People with JavaScript will see what our director wants, and people without will still be able to see all of the content on the page.

Creating the Page

We'll start by creating a basic HTML5 page and we'll add in our "welcome" section which will be the default section displayed to users when they visit the page. Here's the code for the page with the navigation bar and the jump links.

[Download](#) `html5_aria/home.html`

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

AwesomeCo

[Welcome](#) [Services](#) [Contact](#) [About](#)

Welcome

The welcome section

© 2010 AwesomeCo.

[Home](#) [About](#) [Terms of Service](#) [Privacy](#)

Figure 5.1: A mockup of the home page using jQuery to change the main content

```

<title>AwesomeCo</title>
<link rel="stylesheet" href="style.css" type="text/css">
</head>
<body>
  <header id="header">
    <h1>AwesomeCo </h1>
    <nav>
      <ul>
        <li><a href="#welcome">Welcome</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#contact">Contact</a></li>
        <li><a href="#about">About</a></li>
      </ul>
    </nav>
  </header>
  <section id="content"
    role="document" aria-live="assertive" aria-atomic="true">

    <section id="welcome">
      <header>
        <h2>Welcome</h2>
      </header>
      <p>The welcome section</p>
    </section>
  </section>
  <footer id="footer">
    <p>&copy; 2010 AwesomeCo.</p>
    <nav>
      <ul>
        <li><a href="http://awesomeco.com/">Home</a></li>
        <li><a href="about">About</a></li>
        <li><a href="terms.html">Terms of Service</a></li>
        <li><a href="privacy.html">Privacy</a></li>

```

```

        </ul>
      </nav>
    </footer>

</body>
</html>

```

The Welcome section has an ID of welcome which matches the anchor in the navigation bar. We can declare our additional page sections in the same fashion.

[Download](#) html5_aria/home.html

```

<section id="services">
  <header>
    <h2>Services</h2>
  </header>
  <p>The services section</p>
</section>

<section id="contact">
  <header>
    <h2>Contact</h2>
  </header>
  <p>The contact section</p>
</section>

<section id="about">
  <header>
    <h2>About</h2>
  </header>
  <p>The about section</p>
</section>

```

Our four content regions are wrapped by this markup:

[Download](#) html5_aria/home.html

```

<section id="content"
  role="document" aria-live="assertive" aria-atomic="true">

```

The attributes on this line tell screen readers that this region of the page updates.

Polite and Assertive updating

There are two types of methods for alerting the user to changes on the page when using aria-live. The polite method is designed to not interrupt the user's workflow. For example, if the user's screenreader is reading a sentence and another region of the page updates, and the mode is set to polite, then the screenreader will finish reading the current sentence.

However, if the mode was set to `assertive`, then it's considered high priority and the screenreader will stop and begin reading the new content. It's really important that you use the appropriate type of interruption when you're developing your site. Overuse of "assertive" can disorient and confuse your users. Only use `assertive` if you absolutely must. In our case, it's the right choice, as we will be hiding the other content.

Atomic updating

The second parameter, `aria-atomic=true` instructs the screen reader to read the entire contents of the changed region. If we set it to `false`, it would tell the screenreader to only read nodes that changed. We're replacing the entire content, so telling the screenreader to read it all makes sense in this case. If we were replacing a single list item, or appending to a table with AJAX, we would want to use `false` instead.

Hiding Regions

To hide the regions, we need to write a little bit of JavaScript and attach it to our page. We'll create a file called `application.js`, and then we include this file as well as the jQuery library on our page.

[Download](#) `html5_aria/home.html`

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>

<script type="text/javascript"
  charset="utf-8"
  src="javascripts/application.js">
</script>
```

Our `application.js` file contains this simple script:

[Download](#) `html5_aria/javascripts/application.js`

```
Line 1 // HTML5 structural element support for IE 6, 7, and 8
- document.createElement("header");
- document.createElement("footer");
- document.createElement("section");
5 document.createElement("aside");
- document.createElement("article");
-
- $(function(){
-
10   $("#services, #about, #contact").hide().addClass("hidden");
-   $("#welcome").addClass("visible");
-
```

```

-   $("nav ul").click(function(event){
-
15      target = $(event.target);
-      if(target.is("a")){
-          event.preventDefault();
-          if ( $(target.attr("href")).hasClass("hidden") ){
-              $(".visible").removeClass("visible").addClass("hidden").hide();
20              $(target.attr("href")).removeClass("hidden").addClass("visible").show();
-
-          };
-      };
-
25  });
-
-  });

```

On line 10 we hide the "services", "about", and "contact" sections. We also apply a class of "hidden" to them and then on the next line we apply a class of "visible" to the default "welcome" section. Adding these classes makes it really easy to identify which sections need to be turned off and on when we do the toggle.

We capture any clicks to the navigation bar on line 13 and then we determine which element was clicked on 16. If the user clicked a link, we check to see if the corresponding section is hidden. The href attribute of the clicked link can easily help us locate the corresponding section using jQuery selectors, which you can see on line 18.

If it's hidden, we hide everything else and then show the selected section.

That's all there is to it. The screenreaders should detect the region changes.

Falling Back

Like roles, this solution can be used right now by the latest versions of screen readers. By following good practices such as unobtrusive JavaScript, we have a simple implementation that can work for a reasonably wide audience. Older browsers and screen readers will ignore the additional attributes, so there's no danger in adding them to our markup.

The Future

HTML5 and the WIA-ARIA specification have paved the way for a much more accessible web. With the ability to identify changing regions on the page, developers can develop richer JavaScript applications without worrying so much about accessibility issues.

Part II

New Sights And Sounds

Drawing On The Canvas

If you wanted an image in a web application, you'd traditionally open up your graphics software of choice, create an image, and embed it on your page with an `img` tag. If you wanted animations, you'd use Flash. HTML5's `canvas` element lets developers create images and animations in the browser programmatically using JavaScript. We can use the `canvas` to create simple or complex shapes or even create graphs and charts without resorting to server-side libraries, Flash, or other plugins. Coincidentally, we'll do both of those in this chapter.

First, we'll get familiar with how we use JavaScript and the `canvas` element together by drawing some simple shapes as we construct a version of the AwesomeCo logo. Then we'll use a graphing library that's specifically designed to work with the `canvas` to create a bar graph of browser statistics. We'll also discuss some of the special fallback challenges that we'll face due to the fact that the `canvas` is more of a programming interface than an element.

Feature	Use	Supported Browsers
Canvas element	<code><canvas></code> Fallback content <code></canvas></code>	<ul style="list-style-type: none">• Safari 4• Firefox 3.6• Chrome 5• Internet Explorer 9

13

Drawing A Logo

The canvas element is a container element much like the script element. It's a blank slate we can draw on. We define a canvas with a width and height like this:

[Download](#) html5canvasgraph/canvas_simple_drawing.html

```
<canvas id="my_canvas" width="150" height="150">
  Fallback content here
</canvas>
```

Unfortunately, you can't use CSS to control or alter the width and height of a canvas element without distorting the contents, so you need to decide on your canvas dimensions when you declare it.

We use JavaScript to put shapes on the canvas. Even if you provided fallback content to those browsers without the canvas element, you still need to prevent the JavaScript code from trying to manipulate it. Find the canvas by its ID and see if the browser supports the canvas' getContext method.

[Download](#) html5canvasgraph/canvas_simple_drawing.html

```
var canvas = document.getElementById('my_canvas');
if (canvas.getContext){
  var context = canvas.getContext('2d');

}else{
  // do something to show the canvas' hidden contents
  // or let the browser display the text within the <canvas> element.
}
```

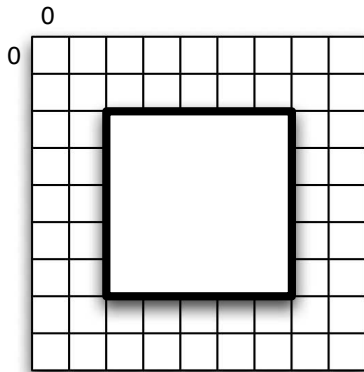
If we get a response from the getContext method, we grab the 2D context for the canvas so we can add objects. If we don't have a context, we need to devise a way to display the fallback content. Since we know that the Canvas element requires JavaScript in order to work we're building a framework to handle fallbacks from the beginning.

Once you have the canvas' context, you simply add elements to that context. To add a red box, you set the fill color and then create the box, like this:

[Download](#) html5canvasgraph/canvas_simple_drawing.html

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
```

The canvas's 2D context is a grid, with the top-left corner as the default origin. When you create a shape, you specify the starting X and Y coordinates and the width and height.



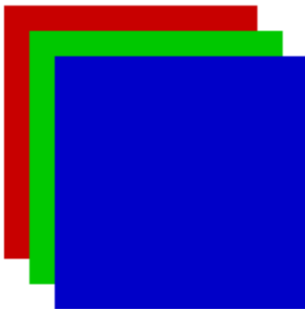
Each shape is added onto its own layer, so you could create three boxes with a ten pixel offset, like this:

[Download](#) html5canvasgraph/canvas_simple_drawing.html

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
context.fillStyle = "rgb(0,200,0)";
context.fillRect (20, 20, 100, 100);
```

```
context.fillStyle = "rgb(0,0,200)";
context.fillRect (30, 30, 100, 100);
```

and they'll stack on top of each other, like this:



Now that you have an understanding of the basics of drawing, let's put together the AwesomeCo logo. It's pretty simple, as you can see from Figure 6.1, on the next page .



Figure 6.1: The AwesomeCo Logo

Drawing The Logo

The logo consists of a string of text, an angled path, a square, and a triangle. Let's start by creating a new HTML5 document, adding a canvas element to the page, and then creating a JavaScript function for drawing the logo which detects whether or not we can use the 2D canvas.

[Download](#) `html5canvasgraph/logo.html`

```
var drawLogo = function(){
  var canvas = document.getElementById('logo');
  var context = canvas.getContext('2d');
};
```

We invoke this method after first checking for the existence of the canvas element, like this:

[Download](#) `html5canvasgraph/logo.html`

```
$(function(){
  var canvas = document.getElementById('logo');
  if (canvas.getContext){
    drawLogo();
  }
});
```

Notice here we're using the jQuery function again to ensure that the event fires when the document is ready. We're looking for an element on the page with the id of logo, so we'd better make sure we add our canvas element to the document so it can be found, and our detection will work.

[Download](#) `html5canvasgraph/logo.html`

```
<canvas id="logo" width="900" height="80">
  <h1>AwesomeCo</h1>
</canvas>
```

Next, let's add the "AwesomeCo" text to the canvas.

Adding Text

Adding text to the canvas involves choosing a font, a font size, an alignment, and then applying the text to the appropriate coordinates on the grid. We can add the text “AwesomeCo” to our canvas like this:

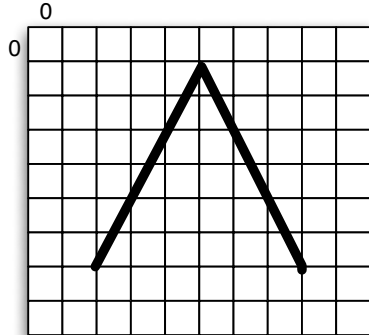
[Download](#) `html5canvasgraph/logo.html`

```
context.font = 'italic 40px sans-serif';
context.textBaseline = 'top';
context.fillText('AwesomeCo', 60, 0);
```

We’re defining the text type and setting its baseline, or vertical alignment before we apply it to the canvas. We’re using the `fillText` method so we get text that’s filled in with the fill color, and we’re setting it 60 pixels to the right so we can make room for the large triangle-shaped path we’ll draw next

Drawing Lines

We draw lines on the canvas by playing a game of “connect-the-dots”. We specify a starting point on the canvas grid, and then specify additional points on the grid to move to. As we move around the canvas, the dots get connected, like this:



We use the `beginPath()` method to start drawing a line, and then we create our path, like this:

[Download](#) `html5canvasgraph/logo.html`

```
context.lineWidth = 2;
context.beginPath();
context.moveTo(0, 40);
context.lineTo(30, 0);
context.lineTo(60, 40 );
context.lineTo(285, 40 );
context.stroke();
context.closePath();
```

When we're all done moving around the canvas, we have to call the `stroke` method to draw the line, and then call the `closePath` method to stop drawing.

Now all that's left is the box and triangle combination that sits within the big triangle.

Moving the Origin

We need to draw a small square and triangle inside of the the larger triangle. When we draw shapes and paths we can specify the X and Y coordinates from the origin at the top left corner of the canvas, but we can also just move the origin to a new location.

Let's draw the smaller inner square by moving the origin.

[Download](#) `html5canvasgraph/logo.html`

```
context.save();
context.translate(20,20);
context.fillRect(0,0,20,20);
```

Notice that before we move the origin we call the `save()` method. This saves the previous state of the canvas so we can revert back easily. It's like a restore point, and you can think of it as a stack. Every time you call `save()` you get a new entry. When we're all done we'll call `restore()` which will restore the top savepoint on the stack.

Now let's use paths to draw the inner triangle, but instead of using a `stroke`, we'll use a `fill`, to create the illusion that the triangle is "cutting into" the square.

[Download](#) `html5canvasgraph/logo.html`

```
context.fillStyle = '#fff';
context.strokeStyle = '#fff';

context.lineWidth = 2;
context.beginPath();
context.moveTo(0, 20);
context.lineTo(10, 0);
context.lineTo(20, 20 );
context.lineTo(0, 20 );
context.fill();
context.closePath();
context.restore();
```

Here we set the `stroke` and `fill` to white (`#fff`) before we begin drawing. Then we draw our lines, and since we moved the origin previously, we're relative to the top-left corner of the square we just drew.

We're almost done, but it needs a little color.

Adding Colors

In Section 13, *Moving the Origin*, on the preceding page, you saw briefly how to set the stroke and fill color for the drawing tools. We could set the color of everything to red just by adding this code before we draw anything:

```
Download html5canvasgraph/logo.html
context.fillStyle = "#f00";
context.strokeStyle = "#f00";
```

But that's a little boring. We can create gradients and assign those to strokes and fills like this:

```
Download html5canvasgraph/logo_gradient.html
// context.fillStyle = "#f00";
// context.strokeStyle = "#f00";
var gradient = context.createLinearGradient(0, 0, 0, 40);
gradient.addColorStop(0, '#a00'); // red
gradient.addColorStop(1, '#f00'); // red
context.fillStyle = gradient;
context.strokeStyle = gradient;
```

We just create a gradient object and set the color stops of the gradient. In this example we're just going between two shades of red, but we could do a rainbow if we wanted¹.

Note that we have to set the color of things before we draw them.

At this point, our logo is complete, and we have a better understanding of how we draw simple shapes on the canvas. However, Internet Explorer 8 and below don't have any support for the canvas element. Let's fix that.

Falling Back

Google released a library called ExplorerCanvas² that makes most of the Canvas API available to Internet Explorer users. All we have to do is include this library on our page.

```
Download html5canvasgraph/logo_gradient.html
<!--[if lte IE 8]>
<script src="javascripts/excanvas.js"></script>
```

-
1. Do *not* do a rainbow, please!
 2. <http://code.google.com/p/explorercanvas/>

<![endif]-->

and things should work just fine in Internet Explorer - but they don't work just yet. At the time of writing, the most stable release doesn't support adding text at all, and the version from their Subversion repository³ doesn't use the correct fonts. Also, there's no support yet for adding gradients on strokes with this library.

So, instead, we rely on other solutions, such as placing a PNG of the logo inside of the canvas element, or we simply don't use the canvas at all. Since this was just an exercise to show you how to draw, it's not the end of the world if we can't use this particular example in a cross-platform production system yet.

3. <http://explorercanvas.googlecode.com/svn/trunk/excanvas.js>

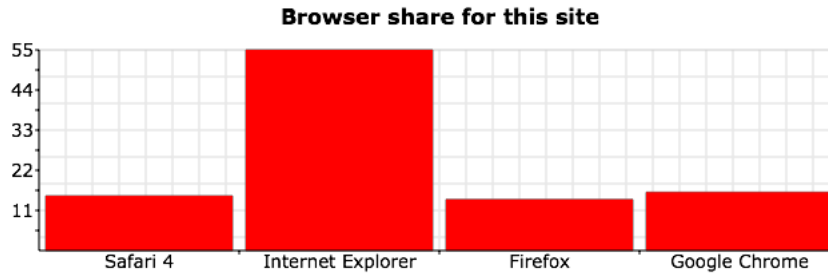


Figure 6.2: A Client-side Bar Graph using the Canvas

14

Graphing Statistics with RGraph

AwesomeCo is doing a lot of work on the website, and senior management would like to see a graph of the web stats. The backend programmers will be able to get the data in realtime, but first they'd like to see if you can come up with a way to display the graph in the browser, so they've provided you with some test data. Our goal is to transform that test data into something that resembles Figure 6.2.

There are lots of ways to draw graphs on a web page. Developers use Flash for graphs all the time, but that has the limitation of not working on some mobile devices like the iPad or iPhone. There are server-side solutions that work well, but those might be too processor-intensive if you're working with realtime data. A standards-based client-side solution like the canvas is a great option as long as we're careful to ensure it works in older browsers. You've already seen how to draw squares, but drawing something complex requires a lot more JavaScript. We need a graphing library to help us along.

The fact that HTML5 isn't available everywhere yet hasn't stopped the developers of the RGraph library⁴. RGraph makes it ridiculously simple to draw graphs using the HTML5 canvas. It's a pure JavaScript solution though, so it won't work for those user agents that don't have

4. <http://www.rgraph.net/>

JavaScript available, but then again, neither will the canvas. Here's the code for a very simple bar graph:

[Download](#) html5canvasgraph/rgraph_bar_example.html

```
<canvas width="500" height="250" id="test">[no canvas support]</canvas>

<script type="text/javascript" charset="utf-8">
  var bar = new RGraph.Bar('test', [50,25,15,10]);
  bar.Set('chart.gutter', 50);
  bar.Set('chart.colors', ['red']);
  bar.Set('chart.title', "A bar graph of my favorite pies");
  bar.Set('chart.labels', ["Banana Creme", "Pumpkin", "Apple", "Cherry"]);
  bar.Draw();
</script>
```

All we have to do is create a couple of JavaScript arrays and the library draws the graph on the canvas for us.

Describing Data with HTML

We could hard code the values for the browser statistics in the JavaScript code, but then only users with JavaScript would be able to see the values. Instead, let's put the data right on the page as text. We can read the data with JavaScript and feed it to the graphing library later.

[Download](#) html5canvasgraph/canvas_graph.html

```
<div id="graph_data">
  <h1>Browser share for this site</h1>
  <ul>
    <li>
      <p data-name="Safari 4" data-percent="15">Safari 4 - 15%</p>
    </li>
    <li>
      <p data-name="Internet Explorer" data-percent="55">Internet Explorer - 55%</p>
    </li>
    <li>
      <p data-name="Firefox" data-percent="14">Firefox - 14%</p>
    </li>
    <li>
      <p data-name="Google Chrome" data-percent="16">Google Chrome - 16%</p>
    </li>
  </ul>
</div>
```

We're using the HTML5 data attributes to store the browser names and the percentages. Although we have that information in the text, it's so much easier to work with programmatically since we won't have to parse strings.

Browser share for this site

- Safari 4 - 15%
- Internet Explorer - 55%
- Firefox - 14%
- Google Chrome - 16%

Figure 6.3: Our Graph as HTML

If you open up the page in your browser, or just look at Figure 6.3, you'll see that the graph data is nicely displayed and readable even without the graph. This will be your fallback content for mobile devices and other users where either the canvas element or JavaScript is not available.

Now, let's turn this markup into a graph.

Turning Our HTML Into A Bar Graph

We're going to use a bar graph, so we'll need to require both the RGraph Bar graph library as well as the main RGraph library. We'll also use jQuery to grab the data out of the document. In the head section of the HTML page, we need to load in the libraries we need.

[Download](#) `html5canvasgraph/canvas_graph.html`

```
<script type="text/javascript"
  charset="utf-8"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
<script src="javascripts/RGraph.common.js" ></script>
<script src="javascripts/RGraph.bar.js" ></script>
```

To build the graph, we need to grab the graph's title, the labels, and the data from the HTML document and pass it to the RGraph library. RGraph takes in arrays for both the labels and the data. We can use jQuery to quickly build those arrays.

[Download](#) `html5canvasgraph/canvas_graph.html`

Line 1 `function drawGraph(){`

```

-   var title = $('#graph_data h1').text();
-
-   var labels = $("#graph_data>ul>li>p[data-name]").map(function(){
5       return $(this).attr("data-name");
-   });
-
-   var percents = $("#graph_data>ul>li>p[data-percent]").map(function(){
-       return parseInt($(this).attr("data-percent"));
10  });
-
-   var bar = new RGraph.Bar('browsers', percents);
-   bar.Set('chart.gutter', 50);
-   bar.Set('chart.colors', ['red']);
15  bar.Set('chart.title', title);
-   bar.Set('chart.labels', labels);
-   bar.Draw();
-
- }

```

First, on line 2 we grab the text for the header. Then on 4 we select all of the elements that have the `data-name` attribute. We use jQuery's `map` function to turn the values from those elements into an array.

We use that same logic again on 8 to grab an array of the percentages.

With the data collected, RGraph has no trouble drawing our graph.

Displaying Alternative Content

In Section 14, *Describing Data with HTML*, on page 113, I could have placed the graph between the starting and ending `canvas` tags. This would hide these elements on browsers that support the `canvas` while displaying them to browsers that don't. However, the content would still be hidden if the user's browser supports the `canvas` element but the user has disabled JavaScript.

We simply leave the data outside of the `canvas` and then hide it with jQuery once we've checked that the `canvas` exists.

[Download](#) `html5canvasgraph/canvas_graph.html`

```

var canvas = document.getElementById('browsers');
if (canvas.getContext){
    $('#graph_data').hide();
    drawGraph();
}

```

With that, our graph is ready, except for people using browsers that don't support the `canvas` element.

Falling Back

When building this solution, we already covered fallbacks for accessibility and lack of JavaScript, but we can create an alternative graph for people who don't have Canvas support but can use JavaScript.

There are a ton of graphing libraries out there, but each one has its own way of grabbing data. Bar graphs are just rectangles with specific heights, and we have all the data on the page we need to construct this graph by hand.

[Download](#) `html5canvasgraph/canvas_graph.html`

```

Line 1
- function drawGraphWithDivs(barColor, textColor, width, spacer, labelHeight){
-   $('#graph_data ul').hide();
-   var container = $("#graph_data");
5
-   container.css( {
-     "display" : "block",
-     "position" : "relative",
-     "height": "300px"}
10  );
-
-   $("#graph_data>ul>li>p").each(function(i){
-
-     var bar = $("<div>" + $(this).attr("data-percent") + "%</div>");
15    var label = $("<div>" + $(this).attr("data-name") + "</div>");
-
-     var commonCSS = {
-       "width": width + "px",
-       "position" : "absolute",
20     "left" : i * (width + spacer) + "px"};
-
-     var barCSS = {
-       "background-color" : barColor,
-       "color" : textColor,
-       "bottom" : labelHeight + "px",
25     "height" : $(this).attr("data-percent") + "%"
-     };
-     var labelCSS = {"bottom" : "0", "text-align" : "center"};
-
-     bar.css( $.extend(barCSS, commonCSS) );
-     label.css( $.extend(labelCSS,commonCSS) );
-
-     container.append(bar);
-     container.append(label);
35  });
-
-  }

```

jQuery CSS versus CSS

In this chapter, we used jQuery to apply styles to the elements as we created them. A lot of that style information, such as the colors of labels and the color of the bars should be offloaded to a separate stylesheet, especially if you want to be able to change the styles independently of the script. For a prototype, this approach is fine, but for a production version, always separate presentation, behavior, and content.

On line 3, we hide the unordered list so that the text values are hidden. We then grab the element containing the graph data and apply some basic CSS styles. We set the positioning of the element to relative on 7, which will let us absolutely position our bar graphs and labels within this container.

Then we loop over the paragraphs in the bulleted list (line 12) and create the bars. Each iteration over the labels creates two div elements; one for the bar itself, and another for the label, which we position below it. So, with just a little bit of math and some jQuery, we are able to recreate the graph. While it doesn't look *exactly* the same, it's close enough to prove the concept.

We then just need to hook it into our canvas detection, like this:

[Download](#) `html5canvasgraph/canvas_graph.html`

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
    $('#graph_data').hide();
    drawGraph();
}
else{
    drawGraphWithDivs("#f00", "#fff", 140, 10, 20);
}
```

You can see the fallback version in Figure 6.4, on the following page. With a combination of JavaScript, HTML, and CSS, we've provided a client-side bar graph and statistical information about browser usage to any platform that requires it. Using the canvas has an additional benefit- it got us to start thinking about a fallback solution from the beginning, rather than trying to wedge something in later. That's really good for accessibility.

Browser share for this site

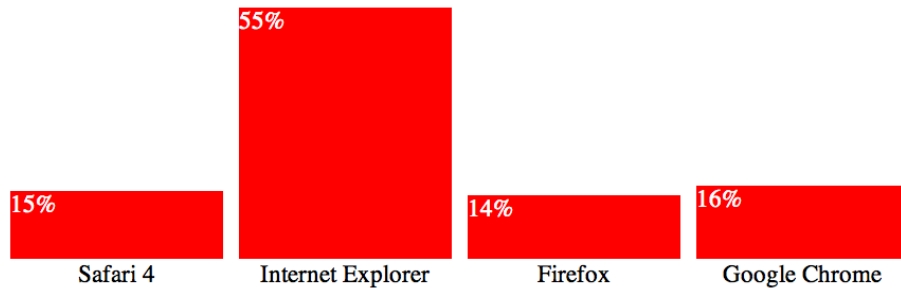


Figure 6.4: Our graph now displays in Internet Explorer



Joe Asks...

Why didn't we try ExplorerCanvas here?

ExplorerCanvas, which we talked about in Section 13, *Falling Back*, on page 110, and RGraph can work really well together. RGraph even bundles a version of ExplorerCanvas in its distribution. However, this combination only works with Internet Explorer 8. If you're working with IE 7 or lower, you'll have to use an alternative solution like the one we discussed. I encourage you to keep an eye on ExplorerCanvas, as it is actively maintained. You might even consider hacking on it yourself to make it work for you.

This is one of the most accessible and versatile methods of graphing data available. You can easily create the visual representation as well as a text-based alternative. This way, everyone can use the important data you're sharing.

The Future

Now that you know a little about how the canvas works, you can start thinking of other ways you might use it. You could use it to create a game, a user interface for a media player, or use it to build a better image gallery. All you need to start painting is a little bit of JavaScript and a little bit of imagination.

Right now, Flash has an advantage over the canvas because it has wider support, but as HTML5 picks up and the canvas is available to a wider audience, more developers will embrace it for simple 2D graphics in the browser. The canvas doesn't require any additional plugins and uses less CPU than Flash, especially on Linux and OSX. Finally, the Canvas provides you a mechanism to do 2D graphics in environments where Flash isn't available. As more platforms support the canvas, you can expect the speed and features to improve, and you'll see more developer tools and libraries appear to help you build amazing things.

But it doesn't stop with 2D graphics. The canvas specification will eventually support 3D graphics as well, and browser manufacturers are implementing hardware acceleration. The canvas will make it possible to create intriguing user interfaces and engaging games using only JavaScript.

Chapter 7

Embedding Audio and Video

Coming soon.

15

Playing Sound Samples with the Audio tag

Coming soon.

16

Building a Cross-Platform Video Tutorial Page

Coming soon.

Chapter 8

Eye Candy

As web developers, we're always interested in making our user interfaces a little more eye-catching, and CSS3 provides quite a few ways for us to do that. We can use our own custom fonts on our pages. We can create elements with rounded corners and drop shadows. We can use gradients as backgrounds, and we can even rotate elements so things don't look so blocky and boring all the time. We can do all of these things without resorting to Photoshop or other graphics programs, and this chapter will show you how. We'll start off by softening up a form's appearance by rounding some corners. Then, we'll construct a prototype banner for an upcoming tradeshow, where we'll learn how to add shadows, rotations, gradients, and opacity. Finally, we'll talk about how to use CSS3's @font-face feature so we can use nicer fonts on the company blog.

Feature	Description	Use	Supported Browsers
border-radius	Rounds corners of elements	border-radius(10px)	<ul style="list-style-type: none"> • Firefox 3 • Safari 3.2 • Chrome 4 • Opera 10.5 • Internet Explorer 9
@font-face	Use specific fonts via CSS	<pre>@font-face { font-family: AwesomeFont; src: url(http://example.com/awesomeco.ttf); font-weight: bold; }</pre>	<ul style="list-style-type: none"> • Firefox 3.5 • Safari 3.2 • Chrome 4 • Opera 10.1 • Internet Explorer 5,6,7,8 (EOT fonts only) • Internet Explorer 9
RGBa support	Use RGB color instead of hex codes along with transparency	background-color: rgba(255,0,0,0.5);	<ul style="list-style-type: none"> • Firefox 3.5 • Safari 3.2 • Chrome 4 • Opera 10.1 • Internet Explorer 9
box-shadow	Drop-shadows on elements	box-shadow: 10px 10px 5px #333	<ul style="list-style-type: none"> • Firefox 3.5 • Safari 3.2 • Chrome 3 • Opera 10.5 • Internet Explorer 9

Rotation Rotate any ele- transform:

Log in

Email
user@example.com

Password
8-10 characters

Log in

Figure 8.1: Our form with round corners

17

Rounding Rough Edges

On the web, everything is a rectangle by default. Form fields, tables, and even sections of web pages all have a blocky, sharp-edged look, and so many designers have turned to different techniques over the years to add rounded corners to these elements to soften up the interface a bit.

CSS3 has support for easily rounding corners, and Firefox and Safari have supported this for quite a long time. Unfortunately, Internet Explorer hasn't jumped on board yet. But we can get around that simply enough.

Softening Up a Login Form

The wireframes and mockups you received for your current project show form fields with rounded corners. Let's round those corners using only CSS3 first. Our goal is to create something that looks like Figure 8.1.

For the login form, we'll use some very simple HTML.

[Download](#) `css3roughedges/rounded_corners.html`

```
<form action="/login" method="post">
  <fieldset id="login">
    <legend>Log in</legend>
    <ol>
```

```

    <li>
      <label for="email">Email</label>
      <input id="email" type="email" name="email">
    </li>
    <li>
      <label for="password">Password</label>
      <input id="password" type="password"
        name="password" value="" autocomplete="off"/>
    </li>
    <li><input type="submit" value="Log in"></li>
  </ol>
</fieldset>
</form>

```

We'll style up the form a bit to give it a slightly better look.

[Download](#) `css3roughedges/style.css`

```

fieldset{
  width: 216px;
  border: none;
  background-color: #ddd;
}

fieldset legend{
  background-color: #ddd;
  padding: 0 64px 0 2px;
}

fieldset>ol{list-style: none;
  padding:0;
  margin: 2px;
}
fieldset>ol>li{
  margin: 0 0 9px 0;
  padding: 0;
}

/* Make inputs go to their own line */
fieldset input{
  display:block;
}

input{
  width: 200px;
  background-color: #fff;
  border: 1px solid #bbb;
}

input[type="submit"]{
  width: 202px;
  padding: 0;
}

```

```
background-color: #bbb;
}
```

These basic styles remove the bullets from the list and ensure that the input fields are all the same size. With that in place, we can apply the rounding effects to our elements.

Browser Specific Selectors

Since the CSS3 specification isn't final, browser makers have added some features themselves and have decided to prefix their own implementations. These prefixes let browser makers introduce features early before they become part of a final specification, and since they don't follow the actual specification, the browser makers can implement the actual specification while keeping their own implementation as well. Most of the time, the vendor-prefixed version matches the CSS specification, but occasionally you'll encounter differences. Unfortunately for you that means you'll need to declare border radius once for each type of browser.

Firefox uses this selector:

[Download](#) `css3roughedges/style.css`

```
-moz-border-radius: 5px;
```

Webkit-based browsers, like Safari and Chrome, use this selector:

[Download](#) `css3roughedges/style.css`

```
-webkit-border-radius: 5px;
```

So, to round all the input fields on our form, we'll need a CSS rule like this:

[Download](#) `css3roughedges/style.css`

```
input, fieldset, legend{
border-radius: 5px;
-moz-border-radius: 5px;
-webkit-border-radius: 5px;
}
```

Add that to your style.css file and you've got rounded corners.

Falling Back

You have everything working in Firefox, Safari, and Google Chrome, but you know it doesn't work in Internet Explorer, and you know it needs to, so you'll need to implement something that gets it as close as possible.

Web developers have been rounding corners for a while now using background images and other techniques, but we're going to keep it as simple as possible. We can detect corner radius with JavaScript and round the corners using any number of rounding techniques. For this example, we'll use jQuery, the jQuery Corner plugin, and a modification of the Corner plugin that rounds text fields.

Detecting Rounded Corners support

Our fallback solution looks very much like the one we used in Section 10, *Falling Back*, on page 86. We'll include the jQuery library and the plugin, detect if the browser supports our attribute, and if it doesn't, we'll activate the plugin. In this case, we need to detect the presence of the border-radius CSS property, but we also need to check for browser-specific prefixes like webkit and moz.

Create corner.js and add this function:

[Download](#) `css3roughedges/corner.js`

```
function hasBorderRadius(){
  var element = document.documentElement;
  var style = element.style;
  if (style){
    return typeof style.borderRadius == "string" ||
      typeof style.MozBorderRadius == "string" ||
      typeof style.WebkitBorderRadius == "string" ||
      typeof style.KhtmlBorderRadius == "string";
  }
  return null;
}
```

We can now detect if our browser is missing support for rounded corners, so let's write the code to do the actual rounding. Thankfully there's a plugin that can get us started.

jQuery Corners

jQuery Corners¹ is a small plugin that rounds corners by wrapping elements with additional div tags and styling them so that the target element looks rounded. However, it doesn't work for form fields, but with a little imagination, we can use this plugin and a little bit of jQuery to make it work.

First, grab jQuery Corners and link to it from your HTML page. While there, also link up your corner.js file.

1. <http://www.malsup.com/jquery/corner/>

Download [css3roughedges/rounded_corners.html](#)

```
<script src="jquery.corner.js" charset="utf-8" type='text/javascript'></script>
<script src="corner.js" charset="utf-8" type='text/javascript'></script>
```

Now we just have to write the code that actually invokes the rounding.

Our formCorners plguin

We're going to write a jQuery plugin so that we can easily apply this rounding to all of the form fields. We already talked about writing jQuery plugins in Section 6, *Falling Back*, on page 57 so I don't need to cover that again. Instead, I'll just walk you through the code for this plugin which is based in part on a solution by Tony Amoyal.²

Add this to your corners.js file:

Download [css3roughedges/corner.js](#)

```
(function($){
    $.fn.formCorner = function(){
        return this.each(function() {
            var input = $(this);
            var input_background = input.css("background-color");
            var input_border = input.css("border-color");
            input.css("border", "none");
            var wrap_width = parseInt(input.css("width")) + 4;
            var wrapper = input.wrap("<div></div>").parent();
            var border = wrapper.wrap("<div></div>").parent();
            wrapper.css("background-color", input_background)
                .css("padding", "1px");
            border.css("background-color", input_border)
                .css("width", wrap_width + "px")
                .css('padding', '1px');
            wrapper.corner("round 5px");
            border.corner("round 5px");
        });
    };
})(jQuery);
```

We're taking a jQuery object which could be an element or a collection of elements and we're wrapping it with two div tags which we then round. We first make the innermost div the same color as the background of the original input. and turn off the border of the actual form field. Then we wrap that field with another field with its own background color, which is the color of the original input's border color, and give it a little bit of padding. This padding is what makes the border's outline visible.

2. <http://www.tonyamoyal.com/2009/06/23/text-inputs-with-rounded-corners-using-jquery-without-image/>

Decide if it's worth the effort

In our example, the client really wanted rounded corners for all browsers. However, you should always keep these kinds of features optional if you can. While some people may argue that there's a real benefit to softening up the way the form looks, you should first have an idea of how many people use browsers that don't support CSS-based rounding. If your visitors are mostly Safari and Firefox users, it may not be worth your time to write and maintain a detection and fallback script.

Imagine two pieces of construction paper, a green one that's four inches wide, and the other a red one that's three inches wide. When you place the smaller one atop the larger one, you'll see a green border around the red one. That's how this works.

Invoking the rounding

With the plugin and our detection library in place, we can now invoke the rounding. Add this to the `corners.js` file:

[Download](#) `css3roughedges/corner.js`

```
Line 1 $(function(){
2     if(!hasBorderRadius()){
3         $("input").formCorner();
4         $("fieldset").corner("round 5px");
5         $("legend").corner("round top 5px cc:#fff");
6     }
7 });
```

We're rounding the three form fields, the fieldset, and finally, on line 5, we're rounding only the top part of the legend, and we're specifying that the cutout of the corner should use white. The plugin uses the background color of the parent for its cutaway color, and that's not appropriate here.

If the browser has support for the `border-radius` property, then the browser runs our plugin. If not, then it'll use the CSS we added earlier.

A minor nudge

IE treats legends a little differently. We can add in a small style fix for IE that pushes the fieldset's legend up a bit so that it looks the same as it does in Firefox and Chrome.

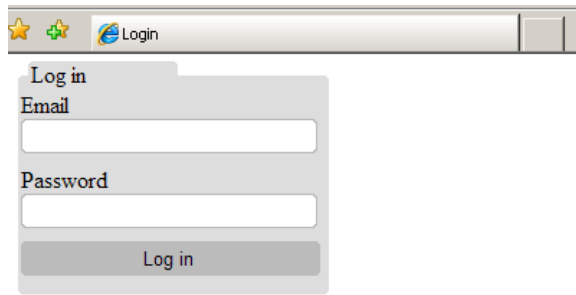


Figure 8.2: Our forms have round corners in Internet Explorer

[Download](#) `css3roughedges/rounded_corners.html`

```
<link rel="stylesheet" href="style.css" type="text/css" media="screen">
<!--[if IE]>
  <style>
    fieldset legend{margin-top: -10px }
  </style>
<![endif]-->
```

Now things look relatively similar on all of the major browsers; you can see the Internet Explorer version in [Figure 8.2](#).

Rounded corners add a bit of softness to your interfaces, and it's extremely easy to use. That said, it's important to be consistent with your use, and to not overuse this technique, just like any other aspect of design.

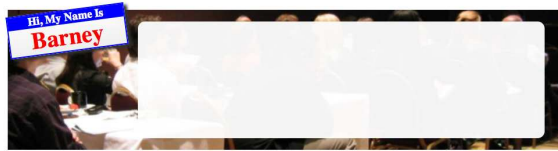


Figure 8.3: The original concept, which we can recreate using CSS3.

18

Working With Shadows, Gradients, and Transformations

While rounded corners get a lot of attention, that's just the beginning of what we can do with CSS3. We can add drop shadows to elements to make them stand out from the rest of the content, we can use gradients to make backgrounds look more defined, and we can use transformations to rotate elements. Let's put several of these techniques together to mock up a banner for the upcoming AwesomeConf, a trade show and conference that AwesomeCo puts on each year. The graphic designer has sent over a PSD that looks like Figure 8.3. We can do the badge, shadow, and even the transparency all in CSS. The only thing we'll need from the graphic designer is the background image of the people.

The Basic Structure

Let's start by marking up the basic structure of the page in HTML.

[Download](#) `css3banner/index.html`

```
<div id="conference">
  <section id="badge">
    <h3>Hi, My Name Is</h3>
    <h2>Barney</h2>
  </section>

  <section id="info">
  </section>
</div>
```

We can style the basics with this:

Download `css3banner/style.css`

```
#conference{
  background-color: #000;
  width: 960px;
  float:left;
  background-image: url('images/awesomeconf.jpg');
  background-position: center;
  height: 240px;
}

#badge{
  text-align: center;
  width: 200px;
  border: 2px solid blue;
}

#info{
  margin: 20px;
  padding: 20px;
  width: 660px;
  height: 160px;
}

#badge, #info{
  float: left;
  background-color: #fff;
}

#badge h2{
  margin: 0;
  color: red;
  font-size: 40px;
}

#badge h3{
  margin: 0;
  background-color: blue;
  color: #fff;
}
```

Once we apply that stylesheet to our page, we have our badge and content region displayed side-by-side as shown in Figure 8.4, on the next page, so let's start styling the badge.

Adding A Gradient

We can add definition to the badge by changing the white background to a subtle gradient that goes from white to light grey. This gradient

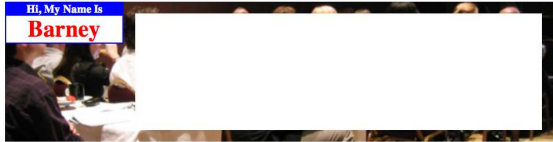


Figure 8.4: Our basic banner

will work in Firefox, Safari, and Chrome, but the implementation is different for Firefox. Chrome and Safari use Webkit's syntax, which was the original proposal, whereas Firefox uses a syntax that's close to the W3C proposal. Once again, we're using browser prefixes, which you saw in Section 17, *Browser Specific Selectors*, on page 127.³

Download `css3banner/style.css`

```
#badge{
  background-image: -moz-linear-gradient(
    top, #fff, #efefef
  );

  background-image: -webkit-gradient(
    linear,left top, left bottom,
    color-stop(0, #fff),
    color-stop(1, #efefef)
  );

  background-image: linear-gradient(
    top, #fff, #efefef
  );
}
```

Firefox uses the `-moz-linear-gradient` method, in which we specify the starting point of the gradient, followed by the starting color, and finally, the ending color. It's simple, but it only lets us use two colors.

Webkit-based browsers let us set color stops. In our example, we only need to go from white to grey, but if we needed to add additional colors, we'd just need to add an additional color stop in the definition.

3. <http://dev.w3.org/csswg/css3-images/#linear-gradients>

Adding a Shadow To The Badge

We can easily make the badge appear to be sitting above the banner by adding a drop shadow. Traditionally, we'd do this shadow in Photoshop by adding it to the image or by inserting it as a background image. However, the CSS3 `box-shadow` property lets us quickly define a shadow on our elements.⁴

We'll apply this rule to our stylesheet to give the badge a shadow:

Download `css3banner/style.css`

```
#badge{
  -moz-box-shadow: 5px 5px 5px #333;
  -webkit-box-shadow: 5px 5px 5px #333;
  -o-box-shadow: 5px 5px 5px #333;
  box-shadow: 5px 5px 5px #333;
}
```

The `box shadow` rule has four parameters. The first is the horizontal offset. A positive number means the shadow will fall to the right of the object, a negative number means it falls to the left. The second parameter is the vertical offset. With the vertical offset, positive numbers make the shadow appear below the box, whereas negative values make the shadow appear above the element.

The third parameter is the blur radius. A value of 0 gives a very sharp value, and a higher value makes the shadow more blurry. The final parameter defines the color of the shadow.

You should experiment with these values to get a feel for how they work, and to find values that look appropriate to you. When working with shadows, you should take a moment to investigate how shadows work in the physical world. Grab a flashlight and shine it on objects, or go outside and observe how the sun casts shadows on objects. This use of perspective is important, because creating inconsistent shadows can make your interface more confusing, especially if you apply shadows to multiple elements incorrectly. The easiest approach you can take is to use the same settings for each shadow you create.

Rotating The Badge

You can use CSS3 transformations to rotate, scale, and skew elements much like you can with vector graphics programs like Flash, Illustrator

4. <http://www.w3.org/TR/css3-background/#the-box-shadow>

Shadows on Text

In addition to adding styles on elements, you can easily apply shadows to your text as well. It works just like box-shadow.

```
h1{text-shadow: 2px 2px 2px #bbbbbb;}
```

You specify the X and Y offsets, the amount of the blur, and the color of the shadow.

IE 6,7, and 8 have support for this as well, using the Shadow filter.

```
filter: Shadow(Color=#bbbbbb,  
  Direction=135,  
  Strength=3);
```

This is the same approach you use to apply a drop shadow to an element.

Shadows on text creates a neat effect, but it can make text harder to read if you make the shadow too strong.

or Inkscape.⁵ This can help make elements stand out a bit more and is another way to make a web page not look so “boxy”. Let’s rotate the badge just a bit so it breaks out of the straight edge of the banner.

Download `css3banner/style.css`

```
#badge{  
  -moz-box-shadow: 5px 5px 5px #333;  
  -webkit-box-shadow: 5px 5px 5px #333;  
  -o-box-shadow: 5px 5px 5px #333;  
  box-shadow: 5px 5px 5px #333;  
}
```

Rotation with CSS3 is pretty simple. All we have to do is provide the degree of rotation and the rendering just works. All of the elements contained within the element we rotate are rotated as well.

Rotating is just as easy as rounding corners, but be careful with it. The ultimate goal of your interface is to make it usable. If you rotate elements containing a lot of content, take care to ensure that the content is easily readable without turning your head too far one direction!

5. <http://www.w3.org/TR/css3-2d-transforms/#transform-property>

Transparent Backgrounds

Graphic designers have used semi-transparent layers behind text for quite some time, and that process usually involves either making a complete image in Photoshop or layering a transparent PNG on top of another element with CSS. CSS3 lets us define background colors with a new syntax that supports transparency.

When you first learn about web development, you learn to define your colors using hexadecimal color codes. You define the amount of red, green, and blue using pairs of numbers. 00 is “all off” or “none” and FF is “all on”. So, the color red would be FF0000 or “all on for red, and all off for blue and all off for green.”

CSS3 introduces the `rgb` and `rgba` functions. The `rgb` function works like the hexadecimal counterpart, but you use values from 0 to 255 for each color. You’d define the color red as `rgb(255,0,0)`.

The `rgba` function works the same way as the `rgb` function, but it takes a fourth parameter to define the amount of opacity, from 0 to 1. If you use 0, you’ll see no color at all, as it’s completely transparent. To make the white box semi-transparent, we’ll add this style rule:

[Download](#) `css3banner/style.css`

```
#info{
  background-color: rgba(255,255,255,0.95);
}
```

When working with transparency values like this, your users’ contrast settings can sometimes impact the resulting appearance, so be sure to experiment with the value and check on multiple displays to ensure you get a consistent result.

While we’re working with the `info` section of our banner, let’s round the corners a bit.

[Download](#) `css3banner/style.css`

```
#info{
  moz-border-radius: 12px;
  webkit-border-radius: 12px;
  o-border-radius: 12px;
  border-radius: 12px;
}
```

With that, our banner looks just pretty good in Safari, Firefox, and Chrome. Now let’s implement a stylesheet for Internet Explorer.

Falling Back

The techniques we used in this section work fine in IE 9, but they're all possible with Internet Explorer 6, 7, and 8 too! We just have to use Microsoft's DirectX filters to pull them off. That means we'll want to rely on a Conditional Comment to load in a specific IE-only stylesheet. We'll also need to use JavaScript to create the section element so we can style it with CSS since these versions of IE don't recognize that element natively.

Download <css3banner/index.html>

```

<!--[if lte IE 8]>

    <script>
    document.createElement("section");
    </script>

    <link rel="stylesheet" href="ie.css" type="text/css" media="screen">

<![endif]-->
</head>
<body>
    <div id="conference">
        <section id="badge">
            <h3>Hi, My Name Is</h3>
            <h2>Barney</h2>
        </section>

        <section id="info">
        </section>
    </div>

</body>
</html>

```

The DirectX filters work in IE 6,7, and 8, but in IE 8 the filters are invoked differently, so you'll be declaring each of these filters twice. Let's start by looking at how we rotate elements.

Rotation

We can rotate elements using these filters, but it's not as easy as just specifying a degree of rotation. To get the effect we want, we need to use the Matrix filter and specify cosines and sines of the angle we want. Specifically, we need to pass the cosine, the negative value of sine, the sine, and the cosine again. ⁶ like this:

6. We're doing a linear transform using a 2x2 matrix.

[Download](#) `css3banner/filters.css`

```
filter: progid:DXImageTransform.Microsoft.Matrix(
  sizingMethod='auto expand',
  M11=0.9914448613738104,
  M12=0.13052619222005157,
  M21=-0.13052619222005157,
  M22=0.9914448613738104
);

-ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
  sizingMethod='auto expand',
  M11=0.9914448613738104,
  M12=0.13052619222005157,
  M21=-0.13052619222005157,
  M22=0.9914448613738104
)";
```

Complicated? Yes, and more so when you look at the above example more closely. Remember that our original angle was *negative* 7.5 degrees. So for our *negative* sine, we need a positive value, and our sine gets a *negative value*..

Math is hard. Let's make gradients instead.

Gradients

IE's Gradient filter works just like the one in the standard, except that you have to type a lot more characters. You provide the starting color and the ending color, and the gradient just shows up.

[Download](#) `css3banner/filters.css`

```
filter: progid:DXImageTransform.Microsoft.gradient(
  startColorStr=#FFFFFF, endColorStr=#EFEFEF
);
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
  startColorStr=#FFFFFF, endColorStr=#EFEFEF
)";
```

Unlike the other browsers, you're applying the gradient directly to the element, rather than to the background-image property.

Let's use this filter again to define the transparent background for our info section.

Transparency

The gradient filter can take extended hexadecimal values for the start and end colors, using the first two digits to define the amount of transparency. We can get very close to the effect we want with this code:

[Download](#) css3banner/filters.css

```
background: none;
filter:
  progid:DXImageTransform.Microsoft.gradient(
    startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
  );

-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
  startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
)";
```

These 8 digit hex codes work very much like the `rgba` function, except that the transparency value comes *first* rather than last. So, we're really looking at alpha, red, green, blue.

We have to remove the background properties on that element to make this work in IE 7. Now, if you've been following along trying to build this stylesheet up, you've noticed that it doesn't actually work yet, but we can fix that.

Putting It All Together

One of the more difficult problems with these IE filters is that we can't define them in pieces. To apply multiple filters to a single element, we have to define the filters as a comma-separated list. Here's what the actual IE stylesheet looks like:

[Download](#) css3banner/ie.css

```
#info{
  background: none;
  filter:
    progid:DXImageTransform.Microsoft.gradient(
      startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
    );
  -ms-filter: "progid:DXImageTransform.Microsoft.gradient(
    startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
  )";
}

#badge{
  filter:
    progid:DXImageTransform.Microsoft.Matrix(
      sizingMethod='auto expand',
      M11=0.9914448613738104,
      M12=0.13052619222005157,
      M21=-0.13052619222005157,
      M22=0.9914448613738104
    ),
    progid:DXImageTransform.Microsoft.gradient(
```

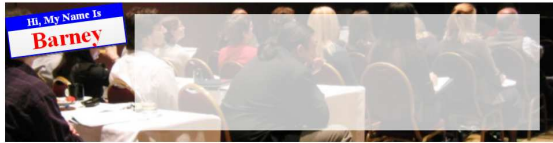


Figure 8.5: Our banner as shown in Internet Explorer 8

```

        startColorStr=#FFFFFF, endColorStr=#EFEFEF
    ),
    progid:DXImageTransform.Microsoft.Shadow(
        color=#333333, Direction=135, Strength=3
    );

-ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
    sizingMethod='auto expand',
    M11=0.9914448613738104,
    M12=0.13052619222005157,
    M21=-0.13052619222005157,
    M22=0.9914448613738104
    ),
    progid:DXImageTransform.Microsoft.gradient(
        startColorStr=#FFFFFF, endColorStr=#EFEFEF
    ),
    progid:DXImageTransform.Microsoft.Shadow(
        color=#333333, Direction=135, Strength=3
    )";
}

```

That's a lot of code to get the desired result, but it shows that it is possible to use these features. If you look at Figure 8.5 you'll see we got pretty close. All we have to do now is round the corners on the info section, and you can refer to *Rounding Rough Edges*, on page 125 to see how to do that.

While these filters are clunky and a little bit quirky, you should still investigate them further in your own projects because you'll be able to provide a similar user experience to your IE users.

Remember that the effects we explored in this section are all presentational. When we created the initial stylesheet, we made sure to apply background colors so that text would be readable. Browsers that cannot understand the CSS3 syntax can still display the page in a readable manner.

Typography is so important to user experience. The book you're reading right now has fonts that were carefully selected by people who understand how choosing the right fonts and the right spacing can make it much easier for people to read this book. These concepts are just as important to understand on the web

The fonts we choose when conveying our message to our readers impacts how our readers interpret that message. Here's a font that's perfectly appropriate for a loud heavy metal band:



But that might not work out so well for the font on the cover of this book:



As you can see, choosing a font that matches your message is really important. The problem with fonts on the web is that we web developers have been limited to a handful of fonts, commonly known as “web-safe” fonts. These are the fonts that are in wide use across most users' operating systems.

To get around that, we've historically used images for our fonts and either directly added them to our page's markup or used other methods like CSS background images or sFIR⁷ which renders fonts using Flash. CSS3's Fonts module offers a much nicer approach.

@font-face

The @font-face directive was actually introduced as part of the CSS2 specification and was implemented in Internet Explorer 5. However, Microsoft's implementation used a font format called Embedded Open

7. <http://www.mikeindustries.com/blog/sifr>

Fonts and Rights

Some fonts aren't free. Like stock photography or other copyrighted material, you are expected to comply with the rights and licences of the material you use on your web site. If you purchase a font, you're usually within your rights to use it in your logo and/or images on your pages. These are called "usage rights". However, the @font-face approach brings a different kind of licensing into play - redistribution rights.

When you embed a font on your page, your users will have to download that font, meaning your site is now distributing this font to others. You need to be absolutely positive the fonts you're using on your pages allow for this type of usage.

Sites like TypeKit* handle the license management for you by hosting the fonts for you in exchange for a monthly fee.

Fonts are just another asset. Pay attention to the license.

*. <http://www.typekit.com>

Type, or EOT, and most fonts out there are in TrueType or OpenType format. Other browsers support the OpenType and TrueType fonts currently.

AwesomeCo's director of marketing has decided that the company should standardize on a font for both print and the web. You've been asked to investigate a font called *Garogier*⁸ which is a simple, thin font that is completely free for commercial use. As a trial run, we'll apply this font to the blog example we created in *Redefining a Blog using Semantic Markup*, on page 23. That way everyone can see the font in action.

Font formats

Fonts are available in a variety of formats, and the browsers your targeting will determine what format you'll need to serve to your visitors.

8. <http://www.fontsqureel.com/fonts/Garogier>

Format	Supported Browsers
Embedded Open-Type(EOT) TrueType (ttf)	<ul style="list-style-type: none"> • Internet Explorer 5,6,7,8
OpenType (otf)	<ul style="list-style-type: none"> • Internet Explorer 9 • Firefox 3.5 • Google Chrome 4.0 • Safari 4
Scalable Vector Graphics (svg)	<ul style="list-style-type: none"> • iOS devices
Web Open Font (woff)	<ul style="list-style-type: none"> • Firefox 3.6 • Internet Explorer 9

Internet Explorer browsers prior to 9 only support a format called Embedded Open Type, or EOT. Other browsers support the more common TrueType and OpenType fonts quite well.

Microsoft, Opera, and Mozilla jointly created the Web Open Font Format which allows lossless compression and better licensing options for font makers.

In order to hit all of these browsers, you have to make your fonts available in multiple formats.

Changing Our Font

The font we're looking at is available at FontSquirrel⁹ in TrueType, WOFF, SVG, and EOT formats, which will work just perfectly.

Using the font involves two steps - defining the font and attaching the font to elements. In the stylesheet for the blog, add this code:

9. You can grab it from <http://www.fontsquirrel.com/fonts/Garogier> and also in the book's downloadable code.



Joe Asks...

How do I convert my own fonts?

If you have developed your own font, or have purchased the rights to a font and need to make it available in multiple formats, the web site FontSquirrel has a converter* you can use that will provide you with the converted fonts as well as a stylesheet with the @font-face code you'll need. Be sure your font's license allows this type of usage though.

*. <http://www.fontsquirrel.com/fontface/generator>

Download [css3fonts/style.css](#)

```
@font-face {
  font-family: 'GarogierRegular';
  src: url('fonts/Garogier_unhinted-webfont.eot');
  src: url('fonts/Garogier_unhinted-webfont.woff') format('woff'),
       url('fonts/Garogier_unhinted-webfont.ttf') format('truetype'),
       url('fonts/Garogier_unhinted-webfont.svg#webfontew0qE009') format('svg');
  font-weight: normal;
}
```

We're defining the font family first, giving it a name, and then supplying the font sources. We're putting the Embedded OpenType version first, so that IE sees it right away, and then we provide the other sources. A user's browser is going to just keep trying sources until it finds one that works.

Now that we've defined the font family, we can use it in our stylesheet. We'll change our original font style so it looks like this:

Download [css3fonts/style.css](#)

```
body{
  font-family: "GarogierRegular";
}
```

With that simple change, our page's text displays in the new font, like the example in Figure 8.6, on the following page.

Applying a font is relatively easy in modern browsers, but we need to consider browsers that don't support this yet.

AwesomeCo Blog!

[Latest Posts](#) [Archives](#) [Contributors](#) [Contact Us](#)

How Many Should We Put You Down For?

Posted by Brian on April 1st, 2010 at 2:39PM

The first big rule in sales is that if the person leaves empty-handed, they're likely not going to come back. That's why you have to be somewhat aggressive when you're working with a customer, but you have to make sure you don't overdo it and scare them away.

"Never
to say :
produc

Figure 8.6: The blog with the new font applied

Falling Back

We've already provided fallbacks for various versions of IE and other browsers, but we still need to ensure our pages are readable in browsers that lack support for the @font-face feature.

We provided alternate versions of the Garogier font, but when we applied the font, we didn't specify any fallback fonts. That means if the browser doesn't support displaying our Garogier font, it's just going to use the browser's default font. That might not be ideal.

Font stacks are lists of fonts ordered by priority. You specify the font you *really* want your users to see first, and then specify other fonts that are suitable fallbacks afterwards.

When creating a font stack, take the extra time to find truly suitable fallback fonts. Letter spacing, stroke width, and general appearance should be similar. The web site UnitInteractive has an excellent article on this¹⁰.

Let's alter our font like this:

[Download](#) css3fonts/style.css

```
font-family: "GarogierRegular", Georgia,
             "Palatino", "Palatino Linotype",
             "Times", "Times New Roman", serif;
```

We're providing a large array of fallbacks here, which should help us maintain a similar appearance. It's not perfect in all cases, but it's bet-

10. <http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks/>

Font Services

Licensing of fonts can get messy, so if you're looking for ways to take advantage of @font-face while avoiding any legal problems, you have a few options.

Typekit* has a large library of fonts available, and they provide tools and code that make it easy to integrate with your web site. They are not a free service, but they are quite affordable if you need to use a specific font.

Google provides the Google Font API†, which is similar to Typekit, but contains only open-source fonts.

There are many other options available as well, but be aware that services like these often require you to use JavaScript to make this work.

*. <http://www.typekit.com/>

†. <http://code.google.com/apis/webfonts/>

ter than relying on the default font which can sometimes be quite hard to read.

Fonts can go a long way to make your page more attractive and easier to read. Experiment with your own work. There are a large number of fonts, both free and commercial, waiting for you.

The Future

In this chapter, we explored a few ways CSS3 replaces traditional web development techniques, but we only scratched the surface. The CSS3 specification talks about 3D transformations and even simple animations, meaning that we can use stylesheets instead of JavaScript to provide interaction cues to users, much like we do with :hover.

In addition, some browsers are already supporting multiple background images, and gradient borders. Finally, keep an eye out for improvements in paged content, like running headers and footers and page number support.

The CSS3 modules, when completed, will make it much easier for us to create richer, better, and more inviting interface elements for our users, so be sure to keep an eye out for new features.

Part III

Beyond HTML5

Working with Client-side Data

Remember when cookies were awesome? Neither do I. Cookies have been rather painful to deal with since they came on the scene, but we have put up with the hassle because they've been the only way to store information on the clients' machines. In order to use them, we have to name the cookie and set its expiration. This involves a bunch of JavaScript code we wrap in a function so we never have to think about how it actually works, kind of like this.

[Download](#) `html5_localstorage/setcookie.js`

```
// via http://www.javascripter.net/faq/settinga.htm
function SetCookie(cookieName,cookieValue,nDays) {
  var today = new Date();
  var expire = new Date();
  if (nDays==null || nDays==0) nDays=1;
  expire.setTime(today.getTime() + 3600000*24*nDays);
  document.cookie = cookieName+"="+escape(cookieValue)
    + ";expires="+expire.toGMTString();
}
```

Aside from the hard-to-remember syntax, there are also the security concerns. Some sites use cookies to track users' surfing behavior, so users disable cookies in some fashion.

HTML5 introduced a few new options for storing data on the client: `LocalStorage`, `SessionStorage`¹, and `Web SQL Databases`². They're easy to use, incredibly powerful, and reasonably secure. Best of all, they're implemented today by several browsers, including Android 2.0 and iOS.

1. <http://www.whatwg.org/specs/web-apps/2007-10-26/#storage>
2. <http://www.whatwg.org/specs/web-apps/2007-10-26/#sql>

Unfortunately, these are no longer part of the HTML5 specification—they've been spun off into their own specifications.

While `LocalStorage`, `SessionStorage`, and Web SQL databases can't replace cookies intended to be shared between the client and the server—like in the case of web frameworks that use the cookies to maintain state across requests—they can be used to store data that only users care about, such as visual settings or preferences. They also come in handy for building mobile applications that can run in the browser but are not connected to the Internet. Many web applications currently call back to a server to save user data, but with these new storage mechanisms, an Internet connection is no longer an absolute dependency. User data could be stored locally and backed up when necessary.

When you combine these methods with HTML5's new offline features, you can build complete database applications right in the browser, that work on a wide variety of platforms, from desktops to iPads and Android phones. In this chapter, you'll learn how to use these techniques to persist user settings and create a simple notes database.

Feature	Description	Supported Browsers
LocalStorage	Stores data in key/value pairs, tied to a domain, and persists across browser sessions	<ul style="list-style-type: none"> • Internet Explorer 8 • Firefox 3 • Safari 4 • Chrome 3 • Opera 10.5 • Android • iOS 3.0
SessionStorage	Stores data in key/value pairs, tied to a domain, and is erased when a browser session ends.	<ul style="list-style-type: none"> • Internet Explorer 8 • Firefox 3 • Safari 4 • Chrome 3 • Opera 10.5 • Android 2.0 • iOS 3.0
Web Databases	SQL Fully relational databases with support for creating tables, inserts, updates, deletes, and selects, with transactions. Tied to a domain and persists across sessions.	<ul style="list-style-type: none"> • Safari 3.2 • Chrome 3 • Opera 10.5 • Android 2.0 • iOS 3.0
Offline Web Applications	Defines files to be cached for offline use, allowing applications to run without an Internet connection.	<ul style="list-style-type: none"> • Firefox 3.5 • Safari 4 • Chrome 4 • Opera 10.6 • Android 2.0 • iOS 3.0

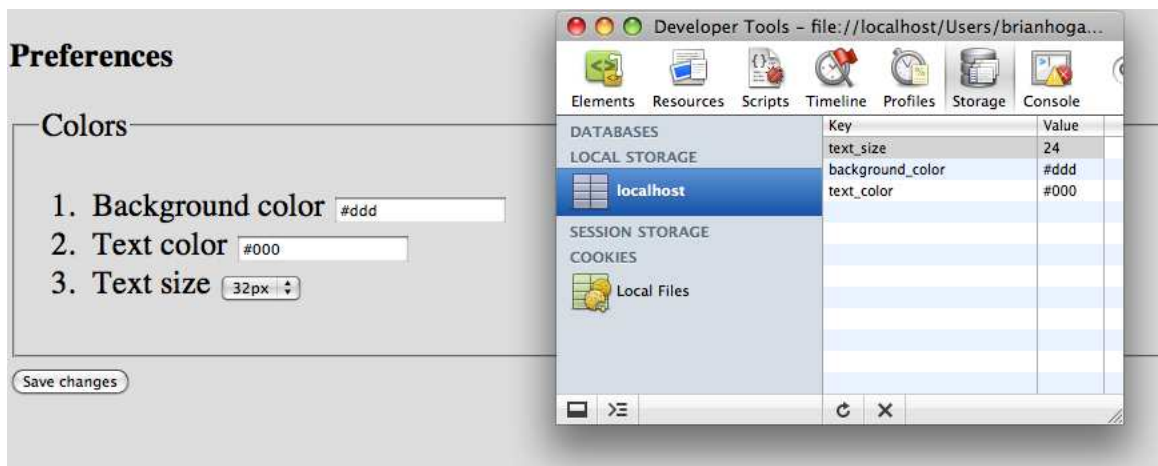


Figure 9.1: Values for the users' preferences are stored in LocalStorage

20

Saving Preferences with LocalStorage

LocalStorage provides a very simple method for developers to persist data on the client's machine. LocalStorage is simply a name-value store built in to the web browser.

LocalStorage persists between browser sessions, and can't be read by other web sites, because it's restricted to the domain you're currently visiting.³

AwesomeCo is in the process of developing a new customer service portal and wants users to be able to change the text size, background, and text color of the site. Let's implement that using LocalStorage, so that when we save the changes, they persist from one browser session to the next. When we're done we'll end up with a prototype that looks like Figure 9.1.

3. Just watch out when you're developing things locally. If you're working on localhost for example, you can easily get your variables mixed up!

Building the Preferences form

Let's craft a form using some semantic HTML5 markup and some of the new form controls you learned about in Chapter 3, *Creating User-friendly Web Forms*, on page 43. We want to let the user change the foreground color, the background color, and adjust their font size.

[Download](#) `html5_localstorage/index.html`

```
<p><strong>Preferences</strong></p>
<form id="preferences" action="add_to_cart" method="post" accept-charset="utf-8">
  <fieldset id="colors" class="">
    <legend>Colors</legend>
    <ol>
      <li>
        <label for="background_color">Background color</label>
        <input type="color" name="background_color" value="" id="background_color">
      </li>
      <li>
        <label for="text_color">Text color</label>
        <input type="color" name="text_color" value="" id="text_color">
      </li>
      <li>
        <label for="text_size">Text size</label>
        <select name="text_size" id="text_size">
          <option value="16">16px</option>
          <option value="20">20px</option>
          <option value="24">24px</option>
          <option value="32">32px</option>
        </select>
      </li>
    </ol>
  </fieldset>

  <input type="submit" value="Save changes">
</form>
```

We'll just use HTML color codes for the color.

Saving and Loading the Settings

To work with the LocalStorage system, you use JavaScript to access the `window.localStorage()` object. Setting a name and value pair is as simple as:

[Download](#) `html5_localstorage/index.html`

```
localStorage.setItem("background_color", $("#background_color").val());
```

Grabbing a value back out is just as easy.

[Download](#) html5_localstorage/index.html

```
var bgcolor = localStorage.getItem("background_color");
```

Let's create a method for saving all of the settings from the form.

[Download](#) html5_localstorage/index.html

```
function save_settings(){
  localStorage.setItem("background_color", $("#background_color").val());
  localStorage.setItem("text_color", $("#text_color").val());
  localStorage.setItem("text_size", $("#text_size").val());
  apply_preferences_to_page();
}
```

Next, let's build a similar method that will load the data from the LocalStorage system and place it into the form fields.

[Download](#) html5_localstorage/index.html

```
function load_settings(){
  var bgcolor = localStorage.getItem("background_color");
  var text_color = localStorage.getItem("text_color");
  var text_size = localStorage.getItem("text_size");

  $("#background_color").val(bgcolor);
  $("#text_color").val(text_color);
  $("#text_size").val(text_size);

  apply_preferences_to_page();
}
```

This method also calls a method that will apply the settings to the page itself, which we'll write next.

Applying the settings

Now that we can retrieve the settings from LocalStorage, we need to apply them to the page. The preferences we're working with are all related to CSS in some way, and we can use jQuery to modify any element's styles.

[Download](#) html5_localstorage/index.html

```
function apply_preferences_to_page(){
  $("body").css("backgroundColor", $("#background_color").val());
  $("body").css("color", $("#text_color").val());
  $("body").css("fontSize", $("#text_size").val() + "px");
}
```

Finally, we need to fire all of this when the document is ready.

Download html5_localstorage/index.html

```
$(function(){
    load_settings();

    $('#form#preferences').submit(function(event){
        event.preventDefault();
        save_settings();
    });
});
```

Falling Back

LocalStorage only works on the latest Internet Explorer, Firefox, Chrome, and Safari, so we'll need a fallback method for older browsers. You have a couple of approaches - save the information on the server, or persist the preferences on the client-side using cookies.

Server-side storage

If you have user accounts in your system, you should consider making the preferences page persist the settings to the user's record in your application. When they log in, you can check to see if any client-side settings exist, and if they don't load them from the server. This way your users keep their settings across browsers and across computers.

To persist to the server, simply ensure your form posts to the server - don't prevent the default submit behavior with JavaScript if there's no support for cookies.

Server-side storage is really the only method that will work if the user disables JavaScript, as you could code your application to fetch the settings from the database and not the LocalStorage hash. Also, this is the only approach you can take if you're storing more than 4 KB of data, since that's the maximum amount of data you can store in a cookie.

Cookies and JavaScript

The tried-and-true combination of cookies and JavaScript can act as a decent fallback. Using the well-known cookie script from Quirksmode,⁴ we can build our own LocalStorage fallback solution.

Detecting LocalStorage support in the browser is pretty simple. We just check for the existence of a localStorage method on the window object:

4. <http://www.quirksmode.org/js/cookies.html>

[Download](#) `html5_localstorage/index.html`

```
if (!window.localStorage){
}
```

Next, we need methods to write the cookies, which we'll borrow from the [Quirksmode](#) article. Add these JavaScript functions to your script block, within the braces:

[Download](#) `html5_localstorage/index.html`

```
function createCookie(name,value,days) {
  if (days) {
    var date = new Date();
    date.setTime(date.getTime()+(days*24*60*60*1000));
    var expires = "; expires="+date.toGMTString();
  }
  else var expires = "";
  document.cookie = name+"="+value+expires+"; path=/";
}

function readCookie(name) {
  var result = ""
  var nameEQ = name + "=";
  var ca = document.cookie.split(';');
  for(var i=0;i < ca.length;i++) {
    var c = ca[i];
    while (c.charAt(0)==' ') c = c.substring(1,c.length);
    if (c.indexOf(nameEQ) == 0){
      result = c.substring(nameEQ.length,c.length);
    }else{
      result = "";
    }
  }
  return(result);
}
```

Finally, we want to make a `LocalStorage` object that uses the cookies as its backend. A very hackish example that just barely makes this work might look like this:

[Download](#) `html5_localstorage/index.html`

```
Line 1 localStorage = (function () {
-   return {
-     setItem: function (key, value) {
-       createCookie(key, value, 3000)
5     },
-
-     getItem: function (key) {
-       return(readCookie(key));
-     }
10  };
```

SessionStorage

We can use LocalStorage for things that we want to persist even after our users close their web browsers, but sometimes we need a way to store some information while the browser is open and throw it away once the session is over. That's where SessionStorage comes into play. It works the same way as LocalStorage but the contents of the SessionStorage are cleared out once the browser session ends. Instead of grabbing the localStorage object, you grab the sessionStorage object.

```
sessionStorage.setItem('name', 'Brian Hogan');
var name = sessionStorage.getItem('name');
```

Creating a fallback solution for this is as simple as ensuring that the cookies you create expire when the browser closes.

```
});
```

Take note of line 4. I'm creating a cookie with an expiration date of 3000 days from now. We can't create cookies that never expire, so I'm setting this to a ridiculously long time into the future.

We've kept the basic implementation of localStorage the same from the outside. If you need to remove items or clear everything out, you'll need to get a little more creative. Hopefully, in the near future, we can remove this hackish solution and rely only on the browser's localStorage() methods.

Storing Data in Client-Side Relational Database

LocalStorage and sessionStorage give us an easy way to store simple name/value pairs on the client's computer, but sometimes we need more than that. HTML5 introduces the ability to store data in relational databases. It's called "Web SQL Storage"⁵ and if you have even a basic background in writing SQL statements, you'll feel right at home in no time. To get you comfortable, we'll use Web SQL Storage to create, retrieve, update, and destroy notes in a client-side database.

CRUD in your Browser

The term CRUD, an acronym for "Create, Retrieve, Update, and Delete"⁶, pretty much describes what we can do with our client-side database. The specification and implementations allow us to insert, select, update, and delete records.

AwesomeCo wants to equip their sales team with a simple application to collect notes while they're on the road. This application will need to let users create new notes, as well as update and delete existing ones. In order to change existing notes, we'll need to let users retrieve them from the database.

Here are the SQL statements we'll need to write in order to make this happen.

Type	Statement
Create a note	<code>INSERT INTO notes (title, note) VALUES("Test", "This is a note");</code>
Retrieve all notes	<code>SELECT id, title, note FROM notes;</code>
Retrieve a specific note	<code>SELECT id, title, note FROM notes where id = 1;</code>
Update a note	<code>UPDATE notes set title = "bar", note = "Changed" where id = 1;</code>
Delete a note	<code>DELETE FROM notes where id = 1;</code>

Figure 9.2: Our notes application interface

The Notes Interface

The interface for the notes application consists of a left-hand sidebar that will have a list of the notes already taken, and a form on the right side with a title field and a larger text area for the note itself. Look at Figure 9.2 to see what we're building.

To start, we need to code up the interface.

[Download](#) `html5sql/index.html`

```
<!doctype html>

<html>
  <head>
    <title>AwesomeNotes</title>
    <link rel="stylesheet" href="style.css">

    <script type="text/javascript"
      charset="utf-8"
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
    </script>
    <!-- EMD:ui -->

    <script type="text/javascript"
      charset="utf-8" src="javascripts/notes.js">
    </script>
```

5. <http://dev.w3.org/html5/webdatabase/>
6. or "Create, Read, Update, and Destroy", if you prefer

```

</head>
<body>
  <section id="sidebar">
    <input type="button" id="new_button" value="New note">
    <ul id="notes">
    </ul>
  </section>

  <section id="main">
    <form>
      <ol>
        <li>
          <input type="submit" id="save_button" value="Save">
          <input type="submit" id="delete_button" value="Delete">

        </li>
        <li><label for="title">Title</label><input type="text" id="title"></li>
        <li><label for="note">Note</label><textarea id="note"></textarea></li>

      </ol>
    </form>
  </section>

</body>
</html>

```

We define the sidebar and main regions using section tags, and we have given IDs to each of the important user interface controls like the save button. This will make it easier for us to locate elements so that we can attach event listeners.

We'll also need a stylesheet so that we can make this look more like the figure. `style.css` looks like this:

[Download](#) `html5sql/style.css`

```

#sidebar, #main{
  display: block;
  float: left;
}

#sidebar{
  width: 25%;
}

#main{
  width: 75%;
}

```



```

form ol{
  list-style: none;
  margin: 0;
  padding: 0;
}

form li{
  padding: 0;
  margin: 0;
}

form li label{
  display:block;
}

#title, #note{
  width: 100%;
  font-size: 20px;
  border: 1px solid #000;
}

#title{
  height: 20px;
}

#note{
  height: 40px;
}

```

This stylesheet turns off the bullet points, sizes the text areas, and lays things out in two columns. Now that we have the interface done, we can build the JavaScript we need to make this work.

Connecting to the Database

We need to make a connection and create a database:

```

Download html5sql/javascripts/notes.js
// Database reference
var db = null;

// Creates a connection to the local database
connectToDB = function()
{
  db = window.openDatabase('awesome_notes', '1.0',
                           'AwesomeNotes Database', 1024*1024*3);
};

```

We're declaring the `db` variable at the top of our script. Doing this makes it available to the rest of the methods we'll create.⁷ We then declare the method to connect to the database by using the `window.openDatabase` method. This takes the name of the database, a version number, a description, and a size parameter.

Creating the Notes Table

Our notes table needs three columns:

Field	Description
<code>id</code>	Uniquely identifies the note. Primary key, integer, autoincrementing
<code>title</code>	The title of the note, for easy reference
<code>Note</code>	The note itself.

Let's create a method to create this table:

[Download](#) `html5sql/javascrpts/notes.js`

```
createNotesTable = function()
{
  db.transaction(function(tx){
    tx.executeSql(
      "CREATE TABLE notes (id INTEGER PRIMARY KEY, title TEXT, note TEXT)", [],
      function(){ alert('Notes database created successfully!'); },
      function(tx, error){ alert(error.message); } );
  });
};
```

We fire the SQL statement inside of a transaction, and the transaction has two callback methods: one for a successful execution, and one for a failure. This is the pattern we'll use for each of our actions.

Note that the `executeSql()` method also takes an array as its second parameter. This array is for binding placeholders in the SQL to variables. This lets us avoid string concatenation and is similar to prepared statements in other languages. In this case, the array is empty as we have no placeholders in our query to populate.

Now that we have our first table, we can make this application actually do something.

7. This puts the variable into the global scope and that's not always a good idea. For this example, we're keeping the JavaScript code as simple as possible.

Loading Notes

When the application loads, we want to connect to the database, create the table if it doesn't already exist, and then fetch any existing notes from the database.

[Download](#) `html5sql/javascrpts/notes.js`

```
// loads all records from the notes table of the database;
fetchNotes = function(){
  db.transaction(function(tx) {
    tx.executeSql('SELECT id, title, note FROM notes', [],
      function(SQLTransaction, data){
        for (var i = 0; i < data.rows.length; ++i) {
          var row = data.rows.item(i);
          var id = row['id'];
          var title = row['title'];

          addToNotesList(id, title);
        }
      });
  });
};
```

This method grabs the results from the database. If it's successful it loops over the results and calls the `addToNotesList` method which we define to look like this:

[Download](#) `html5sql/javascrpts/notes.js`

```
// Adds the list item to the list of notes, given an id and a title.
addToNotesList = function(id, title){
  var notes = $("#notes");
  var item = "<li>";
  item.attr("data-id", id);
  item.html(title);
  notes.append(item);
};
```

We're embedding the ID of the record into a custom data attribute. We'll use that ID to locate the record to load when the user clicks the list item. We then add the new list item we create to the unordered list in our interface with the id of notes. Now we need to add code to load that item into the form when we select a note from this list.

Fetching a specific record

We could add a click event to each list item, but a more practical approach is to watch any clicks on the unordered list and then determine which one was clicked. This way when we add new entries to the list (like

when we add a brand new note) we don't have to add the click event to the list.

Within our jQuery function, we'll add this code:

Download [html5sql/javascripts/notes.js](#)

```
$("#notes").click(function(event){
  if ($(event.target).is('li')) {
    var element = $(event.target);
    loadNote(element.attr("data-id"));
  }
});
```

This fires off the loadNote() method, which looks like this:

Download [html5sql/javascripts/notes.js](#)

```
loadNote = function(id){
  db.transaction(function(tx) {
    tx.executeSql('SELECT id, title, note FROM notes where id = ?', [id],
      function(SQLTransaction, data){
        var row = data.rows.item(0);
        var title = $("#title");
        var note = $("#note");

        title.val(row["title"]);
        title.attr("data-id", row["id"]);
        note.val(row["note"]);
        $("#delete_button").show();
      }
    );
  });
}
```

This method looks a lot like the previous fetchNotes() method. It fires a SQL statement and we then handle the success path. This time, the statement contains a question-mark placeholder and the actual value is in the second parameter as a member of the array.

When we have found a record, we display it in the form. This method also activates the delete button, and embeds the ID of the record into a custom data attribute so that updates can easily be handled. Our "Save" button will check for the existence of the ID. If one exists, we'll update the record. If one is missing, we'll assume it's a new record. Let's write that bit of logic next.

Inserting, Updating, and Deleting records

When a user presses the "Save" button, we'll want to trigger code to either insert a new record or update the existing one. We'll add a click event handler to the "Save" button by placing this code inside of the jQuery function:

Download <html5sql/javascripts/notes.js>

```
$("#save_button").click(function(event){
    event.preventDefault();
    var title = $("#title");
    var note = $("#note");

    if(title.attr("data-id")){
        updateNote(title, note);
    }else{
        insertNote(title, note);
    }
});
```

This method checks the `data-id` attribute of the form's "title" field. If it has no ID, the form assumes we're inserting a new record and invokes the `insertNote` method, which looks like this:

Download <html5sql/javascripts/notes.js>

```
insertNote = function(title, note)
{
    db.transaction(function(tx){
        tx.executeSql("INSERT INTO notes (title, note) VALUES (?, ?)",
            [title.val(), note.val()],
            function(tx, result){
                var id = result.insertId ;
                alert('Record ' + id+ 'saved!');
                title.attr("data-id", result.insertId );
                addToNotesList(id, title.val());
                $("#delete_button").show();
            },
            function(){
                alert('The note could not be saved. ');
            }
        );
    });
};
```

The `insertNote()` method inserts the record into the database and uses the `insertId` property of the resultset to get the ID that was just inserted. We then apply this to the "title" form field as a custom data attribute

and invoke the `addToNotesList()` method to add the note to our list on the side of the page.

Next, we need to handle updates. The `updateNote()` method looks just like the rest of the methods we've added so far:

```
Download html5sql/javascripts/notes.js
updateNote = function(title, note)
{
    var id = title.attr("data-id");
    db.transaction(function(tx){
        tx.executeSql("UPDATE notes set title = ?, note = ? where id = ?",
            [title.val(), note.val(), id],
            function(tx, result){
                alert('Record ' + id + ' updated!');
                $("#notes>li[data-id=" + id + "]").html(title.val());
            },
            function(){
                alert('The note was not updated!');
            }
        );
    });
};
```

When the update statement is successful, we update the title of the note in our list of notes by finding the element with the `data-id` field with the value of the id we just updated.

As for deleting records, it's almost the same. We need a handler for the delete event like this:

```
Download html5sql/javascripts/notes.js
$("#delete_button").click(function(event){
    event.preventDefault();
    var title = $("#title");
    deleteNote(title);
});
```

Then we need the delete method itself, which not only removes the record from the database, but also removes it from the list of notes in the sidebar.

```
Download html5sql/javascripts/notes.js
deleteNote = function(title)
{
    var id = title.attr("data-id");
    db.transaction(function(tx){
        tx.executeSql("DELETE from notes where id = ?", [id],
            function(tx, result){
                alert('Record ' + id + ' deleted!');
            }
        );
    });
};
```

```

        $("#notes>li[data-id=" + id + "]).remove();
    },
    function(){
        alert('The note was not deleted!');
    }
    );
});
};

```

Now we just need to clear out the form so we can create a new record without accidentally duplicating an existing one.

Wrapping Up

Our notes application is mostly complete. We just have to activate the "New" button, which clears the form out when clicked so a user can create a new note after they've edited an existing one. We'll use the same pattern as before - we'll start with the event handler inside of the jQuery function for the "New" button:

[Download](#) `html5sql/javascripts/notes.js`

```

$("#new_button").click(function(event){
    event.preventDefault();
    newNote();
});
//end:newbutton

newNote();

});

```

Next we'll clear out the "data-id" attribute of the "title" field and remove the values from the forms. We'll also hide the delete button from the interface.

[Download](#) `html5sql/javascripts/notes.js`

```

newNote = function(){
    $("#delete_button").hide();
    var title = $("#title");
    title.removeAttr("data-id");
    title.val("");
    var note = $("#note");
    note.val("");
}

```

We should call this newForm method from within our jQuery function when the page loads so that the form is ready to be used. This way the Delete button is hidden too.

That's all there is to it. Our application works on iPhones, Android devices, and desktop machines running Chrome, Safari, and Opera. However, there's little chance this will ever work in Firefox, and it's not supported in Internet Explorer either.

Falling Back

Chrome, Safari, Android, and Mobile Safari let us use this really neat feature, but there's nothing like this available in other browsers. Unlike our other solutions, there are no good libraries available that would let us implement SQL storage ourselves, and so we have no way to provide support to Internet Explorer users natively. However, if this type of application is something you think could be useful, you could convince your users to use Google Chrome, which works on all platforms, for this specific application. That's not an unheard of practice, especially if using an alternative browser allows you to build an internal application that could be made to work on mobile devices as well.

Another alternative is to use the Google Chrome Frame plugin ⁸ Add this to the top of your HTML page right below the head tag:

[Download](#) [html5sql/index.html](#)

```
<meta http-equiv="X-UA-Compatible" content="chrome=1">
```

This snippet gets read by the Google Chrome Frame plugin and activates it for this page.

If you want to detect the presence of the plugin and prompt your users to install it if it doesn't exist, you can add this snippet right above the closing body tag.

[Download](#) [html5sql/index.html](#)

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/chrome-frame/1/CFInstall.min.js">
</script>
```

```
<script>
```

```
  window.attachEvent("onload", function() {
    CFInstall.check({
      mode: "inline", // the default
      node: "prompt"
    });
  });
```

8. <http://code.google.com/chrome/chromeframe/>


```
</script>
```

This will give the user an option to install the plugin so they can work with your site.

Google Chrome Frame may not be a viable solution for a web application meant to be used by the general public, but it works well for internal applications like the one we just wrote. There may be corporate IT policies that prohibit something like this, but I'll leave that up to you to work out how you can get something like this approved if you're in that situation. Installing a plugin is certainly more cost-effective than writing your own SQL database system.

The Future

LocalStorage and SQL databases give developers the ability to build applications in the browser that don't have to be connected to a web server. Applications like the ones we worked on run on an iPad or Android device as well, and so they create the ability to build offline rich applications using familiar tools instead of proprietary platforms. As more browsers enable support, developers will be able to leverage them more, creating applications that run on multiple platforms and devices, that store data locally, and sync up when connected.

The future of Web SQL Storage is unknown. Mozilla has no plans to implement it in Firefox, choosing instead to move forward implementing the IndexedDB specification instead. We'll talk more about that specification in Chapter 11, *Where To Go Next*, on page 174. However, Web SQL Storage has been in use on the iOS devices for a while, and it's likely to stay. This specification could be extremely useful to you if you're developing applications in that space.

Chapter 10

Playing Nicely With Others

22

Cross document Messaging

Coming soon...

23

Getting Chatty with Websockets

Coming soon...

24

Finding Yourself With Geolocation

Coming soon...

Chapter 11

Where To Go Next

Coming Soon...

Chapter 12

jQuery Primer

Coming Soon...

Appendix A

Bibliography

- [HT00] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Reading, MA, 2000.

Index

More Books go here...

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Home Page for HTML5 and CSS3

<http://pragprog.com/titles/bhh5>

Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/bhh5.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)