Learn by doing: less theory, more results

jQuery for Designers

An approachable introduction to web design in jQuery for non-programmers

Beginner's Guide

Natalie MacLees

[PACKT] PUBLISHING open source* community experience distilled

jQuery for Designers

Beginner's Guide

An approachable introduction to web design in jQuery for non-programmers

Natalie MacLees

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

jQuery for Designers
Beginner's Guide

First published: April 2012

Production Reference: 1180412

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84951-670-9

`www.packtpub.com`

Cover Image by Rakesh Shejwal (`shejwal.rakesh@gmail.com`)

Credits

Author
Natalie MacLees

Reviewers
Mark Tapio Kines
Tammy C. Wilson

Acquisition Editor
Sarah Cullington

Lead Technical Editor
Chris Rodrigues

Technical Editors
Manasi Poonthottam
Ankita Shashi
Manali Mehta
Sakina Kaydawala

Copy Editor
Laxmi Subramanian

Project Coordinator
Kushal Bhardwaj

Proofreader
Steve Maguire

Indexers
Hemangini Bari
Tejal Daruwale

Production Coordinator
Prachali Bhiwandkar

Cover Work
Prachali Bhiwandkar

About the Author

Natalie MacLees is a frontend web developer and UI designer. She spends her days on the product team at Geni (`geni.com`), a Los Angeles-based startup that is crowdsourcing the creation of a single family tree of the world. Together with Noel Saw she heads up the Southern California WordPress User's Group (`socalwp.org`), organizing WordPress meetups, help sessions, and workshops. In 2010, she worked as a technical reviewer on WordPress 3 Complete by April Hodge Silver. She makes her online home at `nataliemac.com`.

Her obsession with the Web began when she bought her first computer in 1996 and promptly used it to build her first website. She spends the few moments she manages to be offline each day watching baseball, crafting, reading, baking, bellydancing, collecting Hello Kitty items, and avoiding avocados and olives at all costs. She lives in Los Angeles in eternal hope of developing so many freckles, they'll all touch and make a tan.

Gracious thanks first and foremost to John Resig and the rest of the jQuery team for creating and sharing such a useful and elegant library with the rest of us. Thank you to Houman Allahverdi for being patient while I spent so many hours at the computer, telling him to turn off the TV so I could concentrate. Thank you to Anna Motzer, Mark Tapio Kines, Tammy Wilson, Kimberly Wilkinson, Vanesa Rey, Marlene Angel, Trisha Marcy, Ed Doolittle, LeHang Huynh, Marco Hernandez, McCabe Russell, Teresina Goheen, and Ninno DePatrick for their support, advice, and cheerleading —I couldn't ask for better friends. Thank you to my sisters, Stefanie Elder and Bethany MacLees for being properly impressed that somebody wanted me to write a book. Thank you to my mom, Patricia Demby, and stepfather, John Demby, for being proud of me no matter what. And finally, thank you to Diane Colella Jones for believing in me, even before I did.

About the Reviewers

Mark Tapio Kines has been designing websites professionally since 1995. As a lead designer/art director, his highly diverse client list includes Universal Pictures, Sega, Conan O'Brien, Caltech, and the Internal Revenue Service. He was also the longtime art director for Paramount Pictures' online division in Hollywood. Aside from web design, Mark is an award-winning filmmaker, having written and directed two independent features and several shorts. Today he maintains a busy freelance career as a writer, designer, filmmaker, animator, and creative consultant.

Tammy Wilson has been programming in various languages for many years. She has been creating websites since 2006, when she got tired of the corporate IT world and struck out on her own. Tammy has been focusing on WordPress since Natalie MacLees introduced her to the popular open-source platform. Tammy has a Bachelor of Science degree in Computer Science from California State University, Fullerton.

When not working on websites, she enjoys training and competing with her dogs in agility as well as teaching agility classes. She is also interested in writing iOS Apps.

Table of Contents

Preface

Thank you for reading *jQuery for Designers*. This book is intended for designers who have a basic understanding of HTML and CSS, but want to advance their skill set by learning some basic JavaScript. Even if you've never attempted to write JavaScript before, this book will step you through the process of setting up some basic JavaScript and accomplishing common tasks like collapsing content, drop-down menus, slideshows, and more, all thanks to the jQuery library!

What this book covers

Chapter 1, *Designer, Meet jQuery*, is an introduction to the jQuery library and JavaScript. You'll learn about jQuery's rise to fame, why it's so great for designers, and how it can help you create some fancy special effects without having to learn a lot of code. This chapter also includes a gentle and small introduction to JavaScript, and steps you through writing your first JavaScript code.

Chapter 2, *Enhancing Links*, walks you through some basic enhancements to links. You'll learn how to use jQuery to open a link in a new window, how to add icons to links, and how to turn a list of links into a tabbed interface.

Chapter 3, *Making a Better FAQ Page*, will introduce you to collapsing and showing content, as well as traversing an HTML document to move from one element to another. In this chapter, we'll set up a basic FAQ list, then work on progressively enhancing it to make it easier for our site visitors to use.

Chapter 4, *Building Custom Scrollbars*, is our first look at jQuery plugins. We'll use the jScrollPane plugin to create custom scrollbars that work as expected in several different browsers. We'll take a look at setting up scrollbars, customizing their appearance, and implementing animated scrolling behavior.

Chapter 5, *Creating Custom Tooltips*, takes a look at using the qTip plugin to replace the browser's default tooltips with custom tooltips. Then we take that a step further and create custom tooltips to enhance a navigation bar and use the tooltips to show extra content.

Chapter 6, *Building an Interactive Navigation Menu*, steps you through setting up fully functioning and visually stunning drop-down and fly-out menus. We step through the complex CSS required to get these types of menus working, use the Superfish plugin to fill in features missing from pure CSS solutions, and then take a look at customizing the appearance of the menus.

Chapter 7, *Navigating Asynchronously*, introduces Ajax and shows how to turn a simple website into a single page web app with a bit of jQuery. First, we set up a simple example, then step through a more full-featured example that includes support for incoming links and the back button.

Chapter 8, *Showing Content in Lightboxes*, will step you through showing photos and slideshows in a lightbox using the Colorbox jQuery plugin. Once we get the basics down, we'll also take a look at using the Colorbox plugin to create a fancy login, play a collection of videos, and even set up a single-page website gallery.

Chapter 9, *Creating Slideshows*, walks you through several different approaches to creating image slideshows. First, we step through a basic crossfade slideshow example built from scratch. Then we'll look at using the CrossSlide plugin, the Nivo Slider plugin, and the Galleriffic plugin to create different types of slideshows.

Chapter 10, *Featuring Content in Carousels and Sliders*, introduces carousels, news tickers, and sliders, all built with the help of the jCarousel jQuery plugin. We'll create a horizontal carousel, a vertical news ticker, and a featured content slider. Then, we'll take a look at how plugins can be extended even further when we integrate a slideshow to be used with a carousel.

Chapter 11, *Creating an Interactive Data Grid*, steps you through turning a simple HTML table into a fully-interactive data grid, allowing your site visitors to page through the table, search for entries, and sort by different columns.

Chapter 12, *Improving Forms*, takes a look at how forms can be improved. This chapter walks you through properly setting up an HTML form using some of the latest HTML5 form elements. Then we enhance the form by placing the cursor in the first field, using placeholder text, and validating the site visitor's form entries. Finally, we take a look at the Uniform jQuery plugin which allows us to style even the most stubborn and challenging form elements to achieve a consistent look for our forms across browsers.

What you need for this book

You'll need a text editor for creating HTML, CSS, and JavaScript. Some great free options are TextWrangler for Mac or Notepad++ for Windows. There are also many other options available and you can feel free to use your favorite text editor for any of the examples in this book.

You'll also need a browser. My personal favorite is Google Chrome, which includes some really helpful debugging tools for both CSS and JavaScript. Again, you can feel free to use your favorite browser for the examples in the book.

If you want to create images for your own designs, then Adobe Photoshop and Adobe Illustrator will be helpful, though they are not strictly necessary. Any images needed to set up the examples used in this book are included in the sample code.

Who this book is for

This book is for designers who have the basic understanding of HTML and CSS, but want to extend their knowledge by learning to use JavaScript and jQuery.

Conventions

In this book, you will find several headings appearing frequently.

To give clear instructions of how to complete a procedure or task, we use:

Time for action – heading

1. Action 1
2. Action 2
3. Action 3

Instructions often need some extra explanation so that they make sense, so they are followed with:

What just happened?

This heading explains the working of tasks or instructions that you have just completed.

You will also find some other learning aids in the book, including:

Pop quiz – heading

These are short multiple choice questions intended to help you test your own understanding.

Have a go hero – heading

These set practical challenges and give you ideas for experimenting with what you have learned.

You will also find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " The jQuery object's `filter()` method will allow us to filter a previously selected set of elements."

A block of code is set as follows:

```
$('#tabs a').bind('click', function(e){
   $('#tabs a.current').removeClass('current');
   $('.tab-section:visible').hide();
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
   $(this.hash).show();
   $(this).addClass('current');
   e.preventDefault;
   }).filter(':first').click();
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Some argue that opening a link in a new window breaks the expected behavior of the **Back** button and should be avoided."

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Designer, Meet jQuery

You might have heard quite a lot about jQuery over the past couple of years—it's quickly become one of the most popular code packages in use on the Web today. And you might have wondered what all the fuss was about.

Whether you've tried to figure out JavaScript before and have thrown up your hands in frustration or have been too intimidated to even give it a go, you'll find that jQuery is a wonderfully approachable and relatively easy to learn approach to getting your feet wet with JavaScript.

In this chapter, we will cover:

- What jQuery is and why it's ideal for designers
- Progressive enhancement and graceful degradation
- JavaScript basics
- Downloading jQuery
- Your first jQuery script

What is jQuery?

jQuery is a JavaScript library. That means that it's a collection of reusable JavaScript code that accomplishes common tasks—web developers often find themselves solving the same problems over and over again. Instead of engineering a solution from scratch each time, it makes sense to collect up all those useful bits into a single package that can be included and used in any project. The creators of jQuery have written code to smoothly and easily handle the most common and most tedious tasks we want to accomplish with JavaScript—and they've ironed out all the little differences that need to be worked out to get the code working in different browsers.

It's important to remember that jQuery is JavaScript, not a language of its own. It has all the same rules and is written the same way as JavaScript. Don't let that frighten you away—jQuery really does make writing JavaScript much easier.

jQuery's official tag line is *Write Less, Do More*. That's an excellent and accurate description of the jQuery library—you really can accomplish amazing things in just a few lines of code. My own unofficial tag line for jQuery is *Find Stuff and Do Stuff To It* because finding and manipulating different parts of an HTML document is extremely tedious with raw JavaScript and requires lines and lines of code. jQuery makes that same task painless and quick. Thanks to jQuery, you can not only quickly create a drop-down menu—you can create one that's animated and works smoothly in many different browsers.

Why is jQuery awesome for designers?

So just what is it about jQuery that makes it so easy to learn, even if you have limited or no experience with JavaScript?

Uses CSS selectors you already know

The first thing you'll often do in a jQuery script is select the elements you'd like to work with. For example, if you're adding some effects to a navigation menu, you'll start by selecting the items in the navigation menu. The tools you use for this job are selectors—ways to select certain elements on the page you want to work with.

jQuery borrowed selectors from CSS all the way up through CSS3 and they work even in browsers that don't support CSS3 selectors just yet.

Even though CSS offers a pretty robust set of selectors, jQuery adds a few more, all on its own, to make just the elements you need easy to work with.

If you already know how to do things such as make all the level 1 headings blue or make all the links green and underlined, then you'll easily learn how to select the elements you'd like to modify with jQuery.

Uses HTML markup you already know

If you want to create new elements or modify existing elements with raw JavaScript, you better crack your knuckles and get ready to write lots and lots of code—and it won't make all that much sense, either.

For example, if we wanted to append a paragraph to our page that said *This page is powered by JavaScript*, we would have to first create the paragraph element, then assign the text that should be inside the paragraph to a variable as a string, and finally append the string to the newly created paragraph as a text node. And after all that, we would still have to append the paragraph to the document. Phew! (Don't worry if you didn't understand all of that—it was just to illustrate how much work and code is required to do something this simple.)

With jQuery, adding a paragraph to the bottom of our page is as simple as:

```
$('body').append('<p>This page is powered by jQuery.</p>');
```

That's right—you just append a bit of HTML directly to the body and you're all set. I bet that without understanding JavaScript at all, you can read that line of code and grasp what it's doing. This code is appending a paragraph that reads *This page is powered by jQuery* to the body of my HTML document.

Impressive effects in just a few lines of code

You've got better things to do than to sit and write lines and lines of code to add fade in and fade out effects. jQuery provides you with a few basic animations and the power to create your own custom animations right out of the box. Let's say I wanted to make an image fade into the page:

```
$('img').fadeIn();
```

Yep, that's it—one little line of code in which I select my image and then tell it to fade in. We'll see later in the chapter exactly where this line of code will go in your HTML page.

Huge plugin library available

As I've said earlier, web developers often find themselves solving the same problems over and over again. You're likely not the first person who wanted to build a rotating image slideshow, an animated drop-down menu, or a news ticker.

jQuery has an impressively large library of scripts available freely—scripts to create tooltips, slideshows, news tickers, drop-down menus, date pickers, character counters, and on and on. You don't need to learn how to build all these things from scratch—you just have to learn how to harness the power of plugins. We'll be covering some of the most popular jQuery plugins in this book, and you'll be able to take what you've learned to use any plugin in the jQuery plugin library.

Great community support

jQuery is an open source project—that means it's being collectively built by a team of super-smart JavaScript coders and is freely available for anyone to use. The success or failure of an open source project often depends on the community behind the project—and jQuery has a large and active community supporting it.

That means that jQuery itself is being constantly improved and updated. And on top of that, there are thousands of developers out there creating new plugins, adding features to existing plugins, and offering up support and advice to newcomers—you'll find new tutorials, blog posts and podcasts on a daily basis for just about anything you want to learn.

JavaScript basics

In this section, I am going to cover a few basics of JavaScript that will make things go more smoothly. We're going to look at a little bit of code and work through how it works. Don't be intimidated—this will be quick and painless and then we'll be ready to get on with actually doing something with jQuery.

Progressive enhancement and graceful degradation

There are a few different schools of thought when it comes to enhancing your HTML pages with JavaScript. Let's talk about some of the things we should consider before we dive into the cool stuff.

Progressive enhancement and graceful degradation are essentially two sides of the same coin. They both mean that our page with its impressive JavaScript animations and special effects will still work for users who have less capable browsers or devices. Graceful degradation means that we create our special effect and then make sure it fails gracefully if JavaScript is not enabled. If we take the progressive enhancement approach, we'll first build out a bare bones version of our page that works for everyone, and then enhance it by adding on our JavaScript special effects. I tend to favor the progressive enhancement approach.

Why should we care about users who don't have JavaScript enabled? Well, one of the Web's biggest users—search engines—do not have JavaScript capabilities. When search engines are crawling and indexing your pages, they will not have access to any content that's being loaded into your pages by JavaScript. That's often referred to as dynamic content, and it won't be indexed or found by search engines if it can't be reached with JavaScript disabled.

We're also in an era where we can no longer count on users accessing the web pages we build with a conventional desktop or laptop computer. We're quick to think of smart phones and tablets as the next candidates, and while they are very popular, they still only account for a tiny fraction of Internet access.

People are accessing the Web from gaming consoles, e-book readers, Internet-enabled televisions, a huge variety of mobile devices, and probably hundreds of other ways. Not all of these devices are capable of executing JavaScript—some of them don't even have color screens! Your number one priority should be making sure that your content is available to anyone who asks for it, no matter what device they happen to
be using.

Gotta keep 'em separated

To accomplish this task of making our content available to as wide an audience as possible, we have to think of our web pages in three separate and distinct layers: content, presentation, and behavior.

Content

Content is the meat of our web page—it's the text or the audio or video that we're most interested in presenting on our page, so this is where we start.

Mark up your content with clean, simple HTML code. Use HTML elements the way they were intended to be used. Mark up headings with heading tags, paragraphs with paragraph tags, lists with list tags, and save tables for tabular data.

Browsers have built-in styles for these basic HTML tags—headings will be larger type and probably bolded. Lists will have bullets or numbers. It might not look very fancy, but it's readable and accessible to anyone.

Presentation

The presentation layer is where we start to get fancy. This is where we introduce CSS and start applying our own styles to the content we've created. As we style our page, we might find that we have to go back into our HTML and add some new containers and markup to make things such as multi-column layouts possible, but we should still strive to keep our markup as simple and as straightforward as we can.

Behavior

Once our page has all of our content properly marked up and is styled to look the way we like, now we can think about adding in some interactive behavior. This is where JavaScript and jQuery come in. This layer includes animations, special effects, AJAX, and more.

Designer, meet JavaScript

JavaScript is a powerful and complex language—you can work with it for 10 years and still have more to learn. But don't let that frighten you away, you don't have to know everything about it to be able to take advantage of what it has to offer. In fact, you just have to get down a few basics.

This section introduces some JavaScript basics and JavaScript syntax. Don't be scared away by that developer word—syntax. Syntax just means the rules for writing a language, much like we have rules of grammar for writing English.

Variables

Let's start with something simple:

```
var x = 5;
```

This is a *sentence* in JavaScript. In English, we end a sentence with a period or maybe a question mark or exclamation point. In JavaScript we end our sentences with a semicolon.

In this sentence, we're creating a variable, `x`. A variable is just a container for holding something. In this case, `x` is holding the number `5`.

We can do math with JavaScript like this:

```
var x = 5;
var y = 2;
var z = x + y;
```

Just like algebra, our variable `z` is now holding the value of the number `7` for us.

But variables can hold things other than numbers. For example:

```
var text = 'A short phrase';
```

Here, we've named our variable `text` and it's holding some alphabetical characters for us. This is called a **string**. A string is a set of alpha-numeric characters.

Objects

Objects might be the hardest thing for a newcomer to JavaScript to grasp, but that's often because we overthink it, convinced it has to be more complicated than it actually is.

An object is just what it sounds like—a thing, anything. Just like a car, a dog, or a coffee maker are objects.

Objects have properties and methods. A property is a characteristic of an object. For example—a dog could be tall or short, have pointy ears or floppy ears, and be brown or black, or white. All of these are properties of a dog. A method is something an object can do. For example a dog can run, bark, walk, and eat.

Let's take my dog, Magdelena von Barkington, as an example to see how we would deal with objects, properties, and methods in JavaScript:

```
var dog = Magdelena von Barkington;
```

Here I've created a variable `dog` that I'm using as a container to hold my dog, mostly because I don't want to have to type out her full name each time I refer to her in my code. Now let's say I wanted to get my dog's color:

```
var color = dog.color;
```

I created a container called `color` and I'm using it to hold my dog's color property—`color` is now equal to my dog's color.

Now, I've trained my dog very well, and I would like her to roll over. Here's how I would tell her to roll over with JavaScript:

```
dog.rollOver();
```

`rollOver` is a method—something that my dog can do. After my dog rolls over, I might like to reward her with a treat. Here's how my dog eats a treat with JavaScript:

```
dog.eat('bacon');
```

Wait, what's going on here? Let's take it one step at a time. We have dog, which we know is a container for my dog, Magdelena von Barkington. We have the `eat` method, which we know is something that my dog can do. But my dog can't just eat—she has to eat *something*. We use the parentheses to say what it is that she is eating. In this case, my lucky dog is eating bacon. In JavaScript speak, we would say we were passing bacon to the eat method of the dog.

So you see, objects aren't so difficult—they're just things. Properties are like adjectives—they describe traits or characteristics of an object. Methods are like verbs—they describe actions that an object can do.

Functions

A function is a bit of reusable code that tells JavaScript to do something. For example:

```
function saySomething() {
   alert('Something!');
}
```

This function tells JavaScript to pop up an alert box that says `Something!`. We always start a function with the word `function` then we name our function. That's followed by a set of parentheses and a set of curly brackets. The lines of instruction go inside the curly brackets.

Now, my `saySomething` function won't actually do anything until it's called, so I need to add a line of code to call my function:

```
function saySomething() {
   alert('Something!');
}
saySomething();
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

You might wonder what those parentheses are for. Remember how we could pass things to a method by including them in parentheses?

```
dog.eat('bacon');
```

In this case, we passed bacon to say what the dog was eating. We can do much the same thing for functions. In fact, methods actually are functions—they're just functions specialized for describing what an object can do. Let's look at how we modify our `saySomething` function so that we can pass text to it:

```
function saySomething(text) {
   alert(text);
}
saySomething('Hello there!');
```

In this case, when I wrote the `saySomething` function, I just left a generic container in place. This is called a parameter—we'd say the `saySomething` function takes a text parameter, since I've called my parameter `text`. I chose the name `text` because it's a short and handy descriptor of what we're passing in. We can pass in any bit of text to this function, so `text` is an appropriate name. You can name your parameter anything you'd like—but you'll make your code easier to read and understand if you apply some commonsense rules when you're selecting names for your parameters. A parameter behaves very much like a variable—it's just a container for something.

Downloading jQuery and getting set up

We're ready to include the magic of jQuery into a project, but first we need to download it and figure out how to get it attached to an HTML page. Here, we'll walk through getting a sample HTML file started, and all the associated files and folders we'll need to work through a sample project set up. Once we're finished, you can use this as a template for all the future exercises in the book.

Time for action – downloading and attaching jQuery

Earlier, I described the three layers of an HTML document—content, presentation, and behavior. Let's take a look at how we can set up our files for these three layers:

1. First, let's set up a folder on your hard drive to hold all of your work as you work through the lessons in this book. Find a good place on your hard drive and create a folder called `jQueryForDesigners`.

2. Inside the folder, create a folder called `styles`. We'll use this folder to hold any CSS we create. Inside the `styles` folder, create an empty CSS file called `styles.css`.

   The styles represent our presentation layer. We'll keep all of our styles in this file to keep them separate. Likewise, create a folder called `images` to hold any images we'll use.

3. Next, create a folder called `scripts` to hold our JavaScript and jQuery code. Inside the `scripts` folder, create an empty JavaScript file called `scripts.js`.

   The JavaScript we write here represents our behavior layer. We'll keep all of our JavaScript in this file to keep it separate from the other layers.

4. Now, inside the `jQueryForDesigners` folder, create a new HTML page—very basic as follows:

```
<!DOCTYPE html>
<html>
       <head>
              <title>Practice Page</title>
       </head>
       <body>

              <!-- Our content will go here -->
       </body>
</html>
```

Save this file as `index.html`. The HTML file is our content layer—and arguably the most important layer; as it's likely the reason site visitors are coming to our website at all.

5. Next, we'll attach the CSS and JavaScript files that we made to our HTML page. In the head section, add a line to include the CSS file:

```
<head>
       <title>Practice Page</title>
       <link rel="stylesheet" href="styles/styles.css"/>
</head>
```

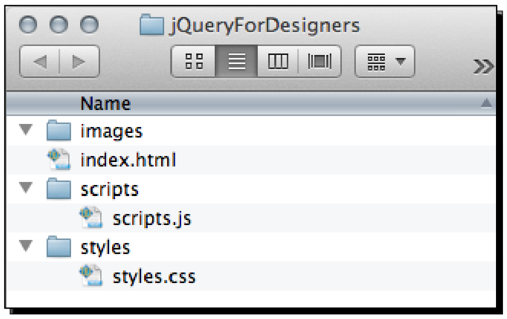
And then head down to the bottom of the HTML file, just before the closing `</body>` tag and include the JavaScript file:

```
       <script src="scripts/scripts.js"></scripts>
       </body>
</html>
```

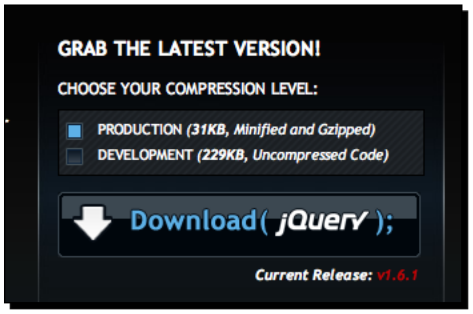
As these files are just empty placeholders, attaching them to your HTML page won't have any effect. But now we have a handy place to write our CSS and JavaScript when we're ready to dive into an exercise.

Note that it's perfectly fine to self-close a `<link>` element, but a `<script>` element always needs a separate closing `</script>` tag. Without it, your JavaScript won't work.

Here's what my folder looks like at this point:

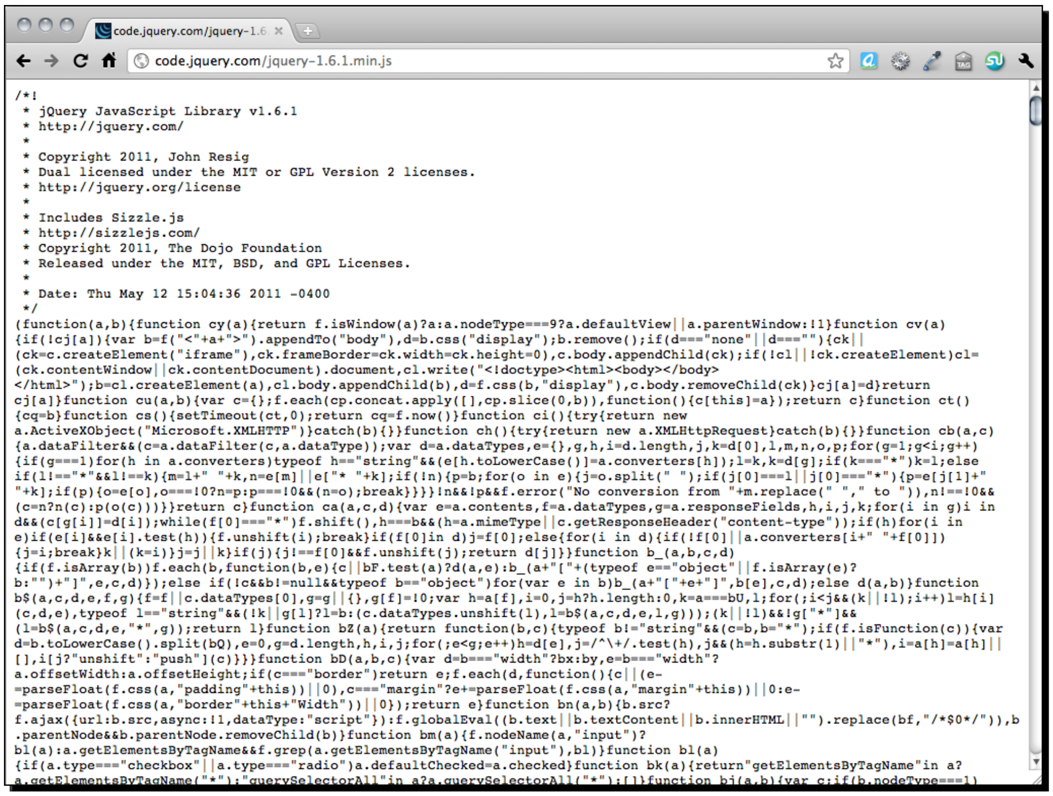


6. Now we have to include jQuery in our page. Head over to `http://jquery.com` and hit the **Download(jQuery);** button:


GRAB THE LATEST VERSION!
CHOOSE YOUR COMPRESSION LEVEL:
PRODUCTION (31KB, Minified and Gzipped)
DEVELOPMENT (229KB, Uncompressed Code)
Download(jQuery);
Current Release: v1.6.1

You'll notice you have two options under **Choose Your Compression Level**. You'll always want to check the **Production** checkbox. This is the version that's ready to use on a website. The **Development** version is for experienced JavaScript developers who want to edit the source code of the jQuery library.

7. Clicking on the **Download** button will open the production jQuery file in your browser window, and it looks a little bit scary, as follows:


code.jquery.com/jquery-1.6
code.jquery.com/jquery-1.6.1.min.js

```
/*!
 * jQuery JavaScript Library v1.6.1
 * http://jquery.com/
 *
 * Copyright 2011, John Resig
 * Dual licensed under the MIT or GPL Version 2 licenses.
 * http://jquery.org/license
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 * Copyright 2011, The Dojo Foundation
 * Released under the MIT, BSD, and GPL Licenses.
 *
 * Date: Thu May 12 15:04:36 2011 -0400
 */
(function(a,b){function cy(a){return f.isWindow(a)?a:a.nodeType===9?a.defaultView||a.parentWindow:!1}function cv(a)
{if(!cj[a]){var b=f("<"+a+">").appendTo("body"),d=b.css("display");b.remove();if(d==="none"||d===""){ck||
(ck=c.createElement("iframe"),ck.frameBorder=ck.width=ck.height=0),c.body.appendChild(ck);if(!cl||!ck.createElement)cl=
(ck.contentWindow||ck.contentDocument).document,cl.write("<!doctype><html><body></body>
</html>");b=cl.createElement(a),cl.body.appendChild(b),d=f.css(b,"display"),c.body.removeChild(ck)}cj[a]=d}return
cj[a]}function cu(a,b){var c={};f.each(cp.concat.apply([],cp.slice(0,b)),function(){c[this]=a});return c}function ct()
{cq=b}function cs(){setTimeout(ct,0);return cq=f.now()}function ci(){try{return new
a.ActiveXObject("Microsoft.XMLHTTP")}catch(b){}}function ch(){try{return new a.XMLHttpRequest}catch(b){}}function cb(a,c)
{a.dataFilter&&(c=a.dataFilter(c,a.dataType));var d=a.dataTypes,e={},g,h,i=d.length,j,k=d[0],l,m,n,o,p;for(g=1;g<i;g++)
{if(g===1)for(h in a.converters)typeof h=="string"&&(e[h.toLowerCase()]=a.converters[h]);l=k,k=d[g];if(k==="*")k=l;else
if(l!=="*"&&l!==k){m=l+" "+k,n=e[m]||e["* "+k];if(!n){p=b;for(o in e){j=o.split(" ");if(j[0]===l||j[0]==="*"){p=e[j[1]+"
"+k];if(p){o=e[o],o===!0?n=p:p===!0&&(n=o);break}}}}!n&&!p&&f.error("No conversion from "+m.replace(" "," to ")),n!==!0&&
(c=n?n(c):p(o(c)))}}return c}function ca(a,c,d){var e=a.contents,f=a.dataTypes,g=a.responseFields,h,i,j,k;for(i in g)i in
d&&(c[g[i]]=d[i]);while(f[0]==="*")f.shift(),h===b&&(h=a.mimeType||c.getResponseHeader("content-type"));if(h)for(i in
e)if(e[i]&&e[i].test(h)){f.unshift(i);break}if(f[0]in d)j=f[0];else{for(i in d){if(!f[0]||a.converters[i+" "+f[0]])
{j=i;break}k||(k=i)}j=j||k}if(j){j!==f[0]&&f.unshift(j);return d[j]}}function b_(a,b,c,d)
{if(f.isArray(b))f.each(b,function(b,e){c||bF.test(a)?d(a,e):b_(a+"["+(typeof e=="object"||f.isArray(e)?
b:"")+"]",e,c,d)});else if(!c&&b!=null&&typeof b=="object")for(var e in b)b_(a+"["+e+"]",b[e],c,d);else d(a,b)}function
b$(a,c,d,e,f,g){f=f||c.dataTypes[0],g=g||{},g[f]=!0;var h=a[f],i=0,j=h?h.length:0,k=a===bU,l;for(;i<j&&(k||!l);i++)l=h[i]
(c,d,e),typeof l=="string"&&(!k||g[l]?l=b:(c.dataTypes.unshift(l),l=b$(a,c,d,e,l,g)));(k||!l)&&!g["*"]&&
(l=b$(a,c,d,e,"*",g));return l}function bZ(a){return function(b,c){typeof b!="string"&&(c=b,b="*");if(f.isFunction(c)){var
d=b.toLowerCase().split(bQ),e=0,g=d.length,h,i,j;for(;e<g;e++)h=d[e],j=/^\+/.test(h),j&&(h=h.substr(1)||"*"),i=a[h]=a[h]||
[],i[j?"unshift":"push"](c)}}}function bD(a,b,c){var d=b==="width"?bx:by,e=b==="width"?
a.offsetWidth:a.offsetHeight;if(c==="border")return e;f.each(d,function(){c||(e-
=parseFloat(f.css(a,"padding"+this))||0),c==="margin"?e+=parseFloat(f.css(a,"margin"+this))||0:e-
=parseFloat(f.css(a,"border"+this+"Width"))||0});return e}function bn(a,b){b.src?
f.ajax({url:b.src,async:!1,dataType:"script"}):f.globalEval((b.text||b.textContent||b.innerHTML||"").replace(bf,"/*$0*/")),b
.parentNode&&b.parentNode.removeChild(b)}function bm(a){f.nodeName(a,"input")?
bl(a):a.getElementsByTagName&&f.grep(a.getElementsByTagName("input"),bl)}function bl(a)
{if(a.type==="checkbox"||a.type==="radio")a.defaultChecked=a.checked}function bk(a){return"getElementsByTagName"in a?
a.getElementsByTagName("*"):"querySelectorAll"in a?a.querySelectorAll("*"):[]}function bj(a,b){var c;if(b.nodeType===1)
```

8. Don't worry, you don't have to read it and you definitely don't have to understand it. Just go to your browser's file menu and choose **Save Page As...** or right-click on the page and select **Save As** and then save the file to your hard drive, inside the `scripts` folder we created. By default, the script will have the version number in the file name. I'm going to go ahead and rename the file to `jquery.js` to keep things simple.

9. Now we just have to include our jQuery script in our page, just like we included our empty JavaScript file. Go to the bottom of your practice HTML file, just before the `<script>` tag we created earlier and add a line to include jQuery:

```
   <script src="scripts/jquery.js"></script>
   <script src="scripts/scripts.js"></script>
</body>
</html>
```

You won't notice any changes to your HTML page—jQuery doesn't do anything on its own. It just makes its magic available for you to use.

Another option for using jQuery

There is nothing wrong with downloading and using your own copy of jQuery, but you do have another option available that can help to improve the performance of your websites. That's to use a CDN-hosted copy of jQuery.

In case you don't know, a **CDN** is a **Content Delivery Network**. The premise behind a CDN is that files download faster from the servers that are physically closer to a site visitor's location. So, for example, if you're in Los Angeles, California, a copy of jQuery that's on a server in Phoenix, Arizona will download faster than a copy that's on a server in New York City. To help this along, a CDN has a copy of the same file on lots of different servers all around the world. Each time a site visitor requests a file, the CDN smartly routes their request to the closest available server, helping to improve response times and overall site performance.

It won't make much of a difference for the relatively simple examples and pages that we'll build in this book, but for a public-facing website, using a CDN-hosted copy of jQuery can make a noticeable difference. There are a few options out there, but the most popular by far is Google's Ajax API CDN. You can get the information on the latest version available and the correct URL at `http://code.google.com/apis/libraries/devguide.html#jquery`.

If you would like to use the Google CDN-hosted version of jQuery in your files, it's as simple as adding the following line of code to your HTML file, instead of the line we used previously to include jQuery:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.
min.js"></script>
```

No downloading the file, no saving your own copy, you can just point your `<script>` tag directly at the copy of jQuery stored on Google's servers. Google will then take care of sending jQuery to your site visitors from the closest available server.

Not only that, but as Google's CDN is so popular, there's a good chance that your site visitor has already visited another site that's also using a Google CDN-hosted copy of jQuery and that they'll have

jQuery cached in their browser. That means that your site visitor won't have to download jQuery at all—it's already saved in their browser and available to be used. How's that for improving performance?

Your first jQuery script

Alright, now that we understand a few basic things about JavaScript and we know how to get our files and folders set up to build a sample exercise, let's build our first simple example page and make it do something fancy with jQuery.

Time for action – getting ready for jQuery

1. Set up your files and folders just like we did in the previous exercise. Inside the `<body>` of the HTML document, add a heading and a paragraph:

   ```
   <body>
     <h1>My First jQuery</h1>
     <p>Thanks to jQuery doing fancy JavaScript stuff is easy.</p>
   </body>
   ```

2. Feel free to create some CSS in the `styles.css` in the styles folder—style this however you would like.

3. Next, open up that empty `scripts.js` file we created earlier and add this bit of script to the file:

   ```
   $(document).ready();
   ```

What just happened?

Let's take this statement one at a time—first a dollar sign? Really? What's that doing in JavaScript?

The `$` here is just a variable—that's all. It's a container for the jQuery function. Remember how I said we might use a variable to save ourselves a few keystrokes? The clever writers of jQuery have provided the `$` variable to save us from having to write out `jQuery` every time we want to use it. This code does the same thing:

```
jQuery(document).ready();
```

Except it takes longer to type. jQuery uses the `$` as its short name because it's unlikely that you'd call a variable `$` on your own since it's an uncommon character. Using an uncommon character reduces the chance that there would be some sort of conflict between some other JavaScript being used on a page and the jQuery library.

So, in this case, we're passing `document` to the jQuery or `$` method, because we want to select our HTML document as the target of our code. When we call the jQuery function, we get a jQuery object. In JavaScript speak, we would say that the jQuery function *returns* a jQuery object. The jQuery object is what gives the jQuery library its power. The entire jQuery library exists to give the jQuery object lots of properties and methods that make our lives easier. We don't have to deal with lots of different sorts of objects—we just have to deal with the jQuery object.

The jQuery object has a method called `ready()`. In this case, the ready method will be called when the document is loaded into the browser, and is ready for us to work with. So `$(document).ready()` just means, "when the document is ready".

Adding a paragraph

So now we're all set to do something when the document is ready, but what is it that we'll do? Let's add a new paragraph to our page.

Time for action – adding a new paragraph

1. We need to tell jQuery what to do when the document is ready. As we want something to happen, we'll pass in a function as follows:

   ```
   $(document).ready(function(){
          // Our code will go here
   });
   ```

   We'll write what's going to happen inside this function.

What about that line that starts with //? That's one way of writing a comment in JavaScript. A // tells JavaScript to ignore everything else on that line because it's a comment. Adding comments to your JavaScript is a great way to help yourself keep track of what's happening on what line. It's also great for helping along other developers who might need to work on your code. It can even be great for helping yourself if you haven't looked at your own code in a few months.

2. Next, we'll add what we want to happen as soon as the document is ready:

```
$(document).ready(function(){
   $('body').append('<p>This paragraph was added with jQuery!</
   p>');
});
```

What just happened?

Our function is using the jQuery function again:

```
$('body')
```

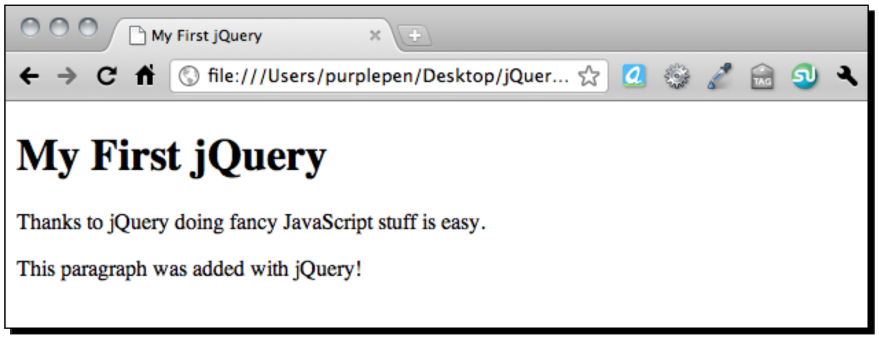
Remember how I said that jQuery uses CSS selectors to find stuff? This is how we use those CSS selectors. In this case, I want the `<body>` tag, so I'm going to pass `'body'` to the jQuery function. This returns the `<body>` tag wrapped in a jQuery object. Handily, the jQuery object has an `append()` method that lets me add something new to the page:

```
$('body').append();
```

All I have to do now is pass the append method the thing I want to add to the page. In quotes, I'll pass in a line of HTML that I'd like to add:

```
$('body').append('<p>This paragraph was added with jQuery!</p>');
```

And that's it! Now when I load my page in a browser, I'll see my heading followed by two paragraphs—jQuery will add the second paragraph as soon as the document is loaded in the browser:



My First jQuery

Thanks to jQuery doing fancy JavaScript stuff is easy.

This paragraph was added with jQuery!

Summary

In this chapter, you were introduced to the jQuery library and learned a few things about it. We covered a bit of JavaScript basics and then we learned how to set up our files and folders for the exercises in this book. Finally, we set up a simple HTML page that took advantage of jQuery to add some dynamic content. Now let's take a look at how we can make links more powerful with jQuery.

2
Enhancing Links

We take links for granted these days, but the truth of the matter is that the humble link is the thing that revolutionized documents and made the Web as we know it today possible. Being able to link a reader directly to another document or to another place within a document had never been possible before.

Because of this, you could say that hyperlinks are the backbone of the Internet—without them search engines wouldn't be possible, nor would most websites. Let's take a look at some ways we can make links work even harder for us.

In this chapter, we will cover:

- How to open links in a new window
- How to add icons to links to identify what type of document we are linking to
- How to turn a list of links into simple tabs

Opening links in a new window

As common as it is to open links in new windows, the practice itself is a little bit controversial. Some argue that the site visitors themselves should decide if they want to open a link in a new window, and many browsers make it easy for them to do just that. Some argue that opening a link in a new window breaks the expected behavior of the **Back** button and should be avoided. Others argue that not opening links in a new window is confusing and disorienting for the site visitors when they suddenly find themselves on a different website.

Wherever you stand on the issue, it's a common request from clients and the practice probably isn't going away any time soon, so it's important to know your options for handling this kind of functionality. I'm going to assume that you're aware of the issues surrounding opening a link in a new window and have carefully weighed all the options and presented an informed argument to your client.

Why not just use the target attribute?

As you may know, HTML makes a `target` attribute available that can be used with links to specify where a link should open. For example, the following code:

```
<a href="http://packtpub.com" target="_new">Link</a>
```

will create a link that will do its best to open in a new window or a new tab, depending on the preferences a user has set in their browser.

The W3C—the body that develops web standards such as HTML—deprecated the use of the `target` attribute for strict document types, but have reintroduced the tag to the HTML5 specification. However, the `target` attribute was intended to be used with frames to control how new pages were loaded into frames and iframes. It was not intended to be used to open a link in a new window for pages that aren't using frames, so strictly speaking, using it for that purpose is incorrect.

Instead, we can use a little bit of JavaScript to create the behavior that we want without using invalid or deprecated code. Let's take a look at how to do that.

Time for action – opening a link in a new window

1. We'll get started with our basic HTML file and associated files and folders that we created in *Chapter 1*, *Designer, Meet jQuery*. Inside the `<body>` of the HTML document, we'll add some links as follows:

   ```
   <h1>Opening Links in a New Window</h1>
   <p>This link will open in a new window: <a href="http://packtpub.
   com">New Window!</a></p>
   <p>This link will not: <a href="http://packtpub.com">Same
   Window!</a></p>
   ```

   This is just a heading and two simple paragraphs, each with a link—one that should open in a new window and one that won't.

2. We need some way to select the link that should open in a new window. This is similar to the situation we would have if we wanted to style one of the links differently from the other with CSS.

If we were using CSS, we could assign the link an `ID` or a `class`. An `ID` would be pretty limiting, as an `ID` must be unique on a page—it would only apply to this one particular link. A `class` would let us style any link that opens in a new window, so that's what we're going to use. Add a `class` to the link that should open in a new window as follows:

```
<a href="http://packtpub.com" class="new-window">New Window!</a>
```

3. Now we can use this class name for both CSS styling and to make the link open in a new window with jQuery. It's a great idea to add an icon to this link—you can add some padding to the left or right side of the link and then add a background image to the link. Open up the empty `styles.css` file inside your `styles` folder and add a bit of CSS as follows:

```
   a.new-window   {
      padding-right: 18px;
      background: url('../images/new-window-icon.png') 100% 50%
no-repeat;
```

4. Next up, we'll open up the `scripts.js` file inside our `scripts` folder, and outside of our document ready statement we'll start off by writing our function to get our `new-window` links and make them open in a new window. Start off by declaring a new function:

```
$(document).ready(function(){

});
     function externalLinks() {
     }
```

Here we've created a new function and named it `externalLinks` as that's a name that makes sense for opening links in new windows. It's always helpful to give your JavaScript functions and variables names that will help you remember what they do.

5. Next, we'll use jQuery to select all the links with the class `new-window`. We'll take advantage of jQuery's CSS selectors to select those links just like we did when we styled them with CSS.

```
function externalLinks()   {
     $('a.new-window');
}
```

6. We've used the `$` shortcut for the jQuery function and passed the CSS selector to the function. It's important to remember to wrap the CSS selector in either single quotes or double quotes. We don't want the link to open a new window until the user clicks on the link, so our next step is to tell the link to run a function when it's clicked on. jQuery makes this very easy. We can use the `bind()` method provided by jQuery to bind a function to the link that will be called when the link is clicked. That will look like this:

   ```
   function externalLinks()   {
      $('a.new-window').bind('click', function() {
      });
   }
   ```

   This bit of code binds a function to our link—when our link is clicked, any code we write inside this new function will be called. But so far, our function is empty and doesn't actually do anything.

7. What we need to do next is get the location the link is sending us to:

   ```
   function externalLinks()   {
      $('a.new-window').bind('click', function() {
         var location = $(this).attr('href');
      });
   }
   ```

   Let's examine this new line of code one bit at a time. First, we've declared a new variable named `location`. As you remember, a variable is just a container. So we've got a new empty container, so now let's look at what we've put inside our container.

   `$(this)` is the jQuery way of referring to the jQuery object that we're currently working with. In this case, we're selecting all links with a class of `new-window` and we've attached this function to be called whenever a site visitor clicks the link. When a site visitor clicks a link, we want to examine the link that was clicked to get the location of where the link is going. A simple and quick way of referring to the current link is to use `$(this)`.

   Next we use the `attr()` method to get an attribute of the link. The location where a link is heading is contained in the `href` attribute, so we pass `href` to the `attr()` method.

   So our container that we've named `location` now contains the URL where the link is pointing, or in this particular case, `http://packtpub.com`.

8. Now that we know where we want to go, all we have to do is open that location in a new window. Opening a new window in JavaScript is simple and straightforward:

```
function externalLinks()   {
   $('a.new-window').bind('click', function() {
      var location = $(this).attr('href');
          window.open(location);
   });
}
```

`window` is a global object in JavaScript that is always available to us. The window object has an `open()` method, and we just have to pass a location to that method so that the browser knows what location to open in a new window.

9. Now, if you open this HTML page in a browser and try clicking the links, you might be disappointed to see that our link does not open in a new window. It's like our JavaScript isn't even on the page at all. We've written a very nice function, but it's not working. That's because functions don't do anything until we tell them to. Telling a function to do its thing in JavaScript speak is 'calling the function'.

We would like this function to fire up, find all the links with the class `new-window`, and bind our new window function to them as soon as the page is loaded in the browser window. That way, our links that should open in new windows will be ready to fire off a new window as soon as our site visitor clicks on one of them.

We just have to add a line inside our document ready statement to call our function:

```
$(document).ready(function(){
     externalLinks();
});

function externalLinks()   {
   $('a.new-window').bind('click', function() {
      var location = $(this).attr('href');
      window.open(location);
   });
}
```

This new bit of code will call our `externalLinks` function as soon as the page loads up in the browser.

10. There's just one thing left to do. Right now if you load the page in a browser and click on a link, you'll find that the link does indeed open in a new window, but it also opens in the current window—so we end up with our new page loaded into two different windows. Not exactly what we had in mind. What we need to do is cancel the default behavior of the link—we've already taken care of opening the location in a new window, so now we need to tell the browser that it can take a break and it doesn't need to do anything when the site visitor clicks on the link. So let's add a parameter to our function and a line of code to cancel the default link behavior.

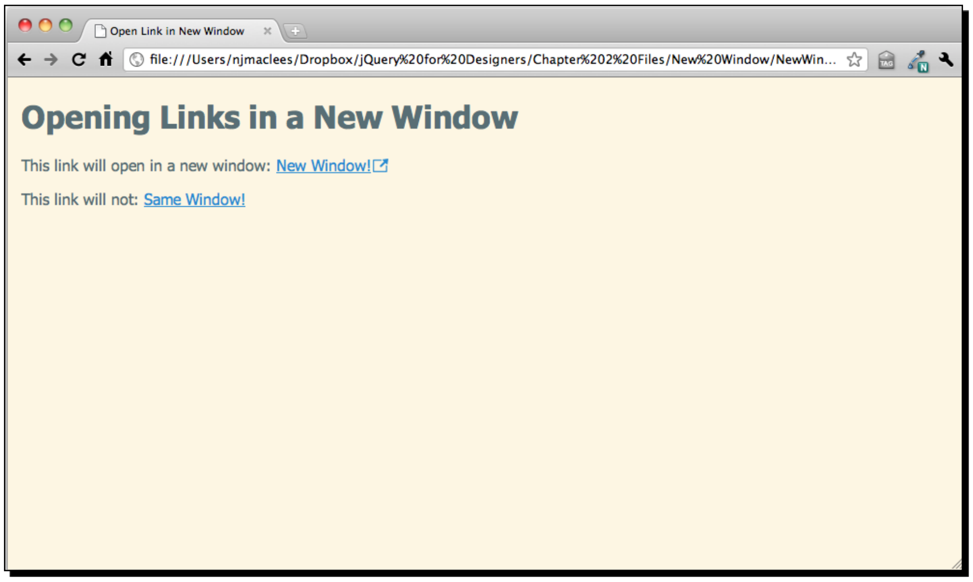
```
function externalLinks()   {
      $('a.new-window').bind('click', function(e) {
      var location = $(this).attr('href');
      window.open(location);
      e.preventDefault();
   });
}
```

You'll notice that the function we've attached to the click action on the link now has an `e` inside the parentheses. This is a parameter that we're passing to this function. In this case `e` represents the click event of the link.

The line of code we add to the function is:

```
e.preventDefault();
```

This tells the browser to stop the default behavior of the link. If you reload the page in your browser and click on the link, you'll see that it correctly opens the destination page in a new window, and it no longer opens the link in the current window:


Open Link in New Window
file:///Users/njmaclees/Dropbox/jQuery%20for%20Designers/Chapter%202%20Files/New%20Window/NewWin...

Opening Links in a New Window

This link will open in a new window: New Window!

This link will not: Same Window!

11. Now, what do you think will happen if we have a second link on the page that should open in a new window? Let's go back to the `<body>` of the document and add a second link that should open in a new window. After the other links, add a new paragraph and link to a new page:

    ```
    <p>This paragraph will open in a new window too: <a href="http://
    nataliemac.com" class="new-window">New Window!</a></p>
    ```

    Be sure to add the `new-window` class to your link.

Now, when you refresh the page in the browser, the new link appears on the page. Try clicking it and you'll see that it opens in a new window too, just like the other `new-window` link.

What just happened?

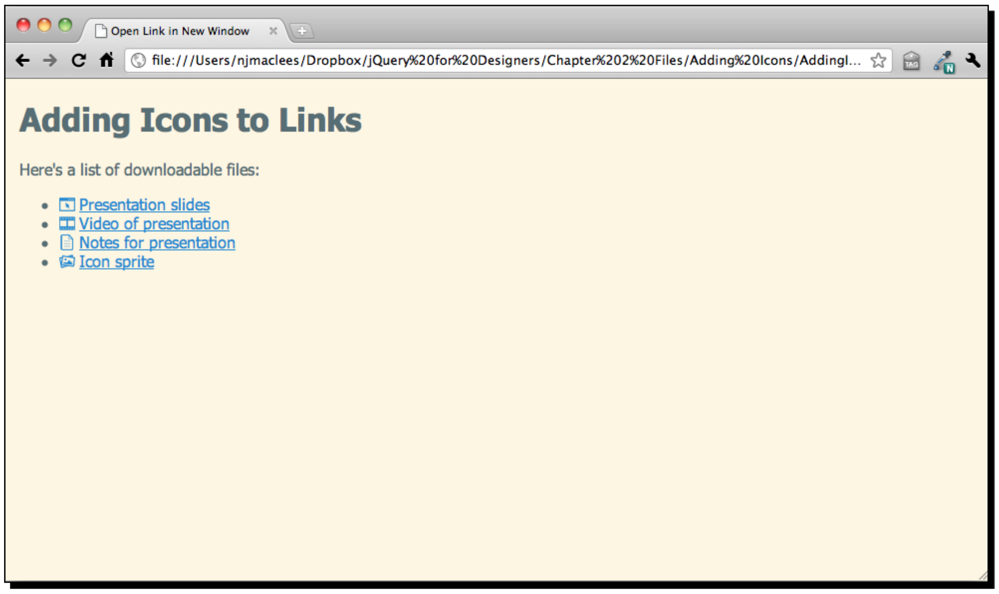
We added a CSS class name to the links that we wanted to open in a new window. Now, any link we create on our page with the `new-window` class will open in a new window—but how does JavaScript know which page to open in a new window when there are multiple links?

The answer lies in our `externalLinks` function. We selected all links with the `new-window` class and bound a function to fire when those links were clicked. Inside that function, we captured the link's location. This function doesn't run until a link is clicked. Until then, it's just sitting on the sidelines, waiting to be called into action. When a link with the `new-window` class is clicked, our function springs into action, capturing the location of that specific link and opening up a new window pointed at that link's location.

Adding icons to links

Adding icons to links is one of the simplest ways to communicate the link type to your site visitor. You might have different icons for different sections of your site, or you might want to provide some downloadable files to your site visitors—for example, a PDF or e-book that you've written, the slides for a presentation you gave, or some stock icons or photography that you've created. Adding icons to these types of links can help give a visual clue to your site visitors so they know what to expect when they click on the link. Let's take a look at how we can add appropriate icons to different link types with jQuery.

Here's an example of what our page will look like after we've added icons to our links:


Open Link in New Window
file:///Users/njmaclees/Dropbox/jQuery%20for%20Designers/Chapter%202%20Files/Adding%20Icons/AddingI...

Adding Icons to Links

Here's a list of downloadable files:

- Presentation slides
- Video of presentation
- Notes for presentation
- Icon sprite

Time for action – creating a list of links

1. We'll get started with our basic HTML file and associated folders, like we created in *Chapter 1*, *Designer, Meet jQuery*. We'll add a list of links to a few different types of downloadable files to the `<body>` of the HTML document:

```
<h1>Adding Icons to Links</h1>
<p>Here's a list of downloadable files:</p>
<ul>
   <li><a href="presentation.ppt">Presentation slides</a></li>
   <li><a href="video.mp4">Video of presentation</a></li>
   <li><a href="notes.pdf">Notes for presentation</a></li>
   <li><a href="icons.gif">Icon sprite</a></li>
</ul>
```

When we view this list in a browser, we'll see a bulleted list of links—there's no visual indication what type of file lies behind each link—the user has to guess based on the text of the link. Let's get all of our links and add an appropriate class name to each one based on which file type the link is pointing to. To do this, we'll use jQuery's attribute selectors.

2. Next up, we'll get ready to add our JavaScript to our page. Open up the `scripts.js` file inside the `scripts` folder.

   Let's figure out how we can distinguish one type of link from another. The `<a>` link has an `href` attribute. That `href` attribute tells us the URL of the page or file the link is taking us to, but it also gives us the information that we need to select links with different values in that attribute. Let's take a look at how jQuery attribute selectors work:

   ```
   $('a')
   ```

   This will select all of the links on the page. If we wanted to get only the `<a>` tags with an `href` attribute, we could modify our selector as follows:

   ```
   $('a[href]')
   ```

   We could take that a step further and get only the links where the attribute was equal to a certain value:

   ```
   $('a[href="video.mp4"]')
   ```

   This selector is only going to select the link that links to the `video.mp4` file. Note the way that single and double quotes are nested here—I can use either single or double quotes to wrap my selector, but if I need to quote something inside my selector, I have to be careful to choose the other type of quotes.

   We want to add a class name to each of these links so that we can style them with CSS to add our icon as a background image. To do that, we'll use the `.addClass()` method of the jQuery object. Using what we've learned so far, we could do something like this inside of our document ready statement:

   ```
   $(document).ready(function(){
     $('a[href="presentation.ppt"]').addClass('presentation');
     $('a[href="video.mp4"]').addClass('video');
     $('a[href="notes.pdf"]').addClass('pdf');
     $('a[href="icons.gif"]').addClass('image');
   });
   ```

   ...but that's not very flexible. What if we wanted to add a second video or another PDF file? We would have to adjust our jQuery to match. Instead, let's make our links a little more flexible by simply looking at the file extension of the link's `href` attribute. jQuery will allow us to check if an attribute begins with certain characters, ends with certain characters, or contains certain characters. You can get the full list of possible attribute selectors in the jQuery documentation at `http://api.jquery.com/category/selectors/`.

To check if an attribute begins with certain characters, use `^=` as follows:

```
$('a[href^="video"]')
```

To check if an attribute contains certain characters anywhere in the name, use `*=` as follows:

```
$('a[href*="deo"]')
```

In this case, the file extension is always the last part of the link, so we'll use the ends with attribute selector, which uses `$=` as follows:

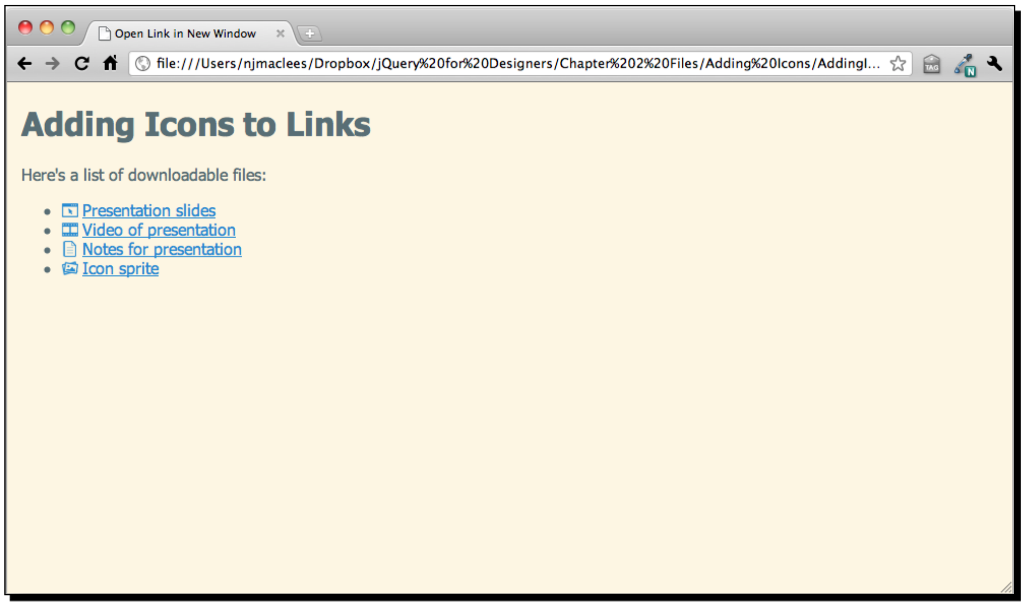
```
$(document).ready(function(){
   $('a[href$="ppt"]').addClass('presentation');
   $('a[href$="mp4"]').addClass('video');
   $('a[href$="pdf"]').addClass('pdf');
   $('a[href$="gif"]').addClass('image');
});
```

3. Now, any links we add with a `.pdf` extension, for example, will automatically have the `pdf` class given to them. If you refresh the page in a browser at this point, you won't see any difference in the page, but if you inspect the **DOM** (**Document Object Model**) using a browser inspection tool such as the ones built into Chrome and WebKit or Firebug for Firefox, you'll see that the links have the class names assigned to them. All that's left to do is to write the CSS to include the icons. Open up the `styles.css` file inside the `styles` folder and add some lines of code as follows:

```
a   {
   background: 0 50% no-repeat;
   padding-left: 20px;
   }

a.presentation   {
   background-image: url(../images/presentation.gif);
   }

a.video   {
   background-image: url(../images/video.gif);
   }

a.pdf   {
   background-image: url(../images/pdf.gif);
   }

a.image   {
   background-image: url(../images/image.gif);
   }
```

You'll have to make sure that you place your icon images inside the `images` folder. You can use the icon images included with the sample code for this chapter or create your own.

Now, if you refresh the page in the browser, you'll see each of the links show the appropriate icon. If you add new links to these four file types to your page, they'll all have the icons as well. We've created a flexible and easy solution for adding icons to links.


Open Link in New Window
file:///Users/njmaclees/Dropbox/jQuery%20for%20Designers/Chapter%202%20Files/Adding%20Icons/AddingI...

Adding Icons to Links

Here's a list of downloadable files:

- Presentation slides
- Video of presentation
- Notes for presentation
- Icon sprite

What just happened?

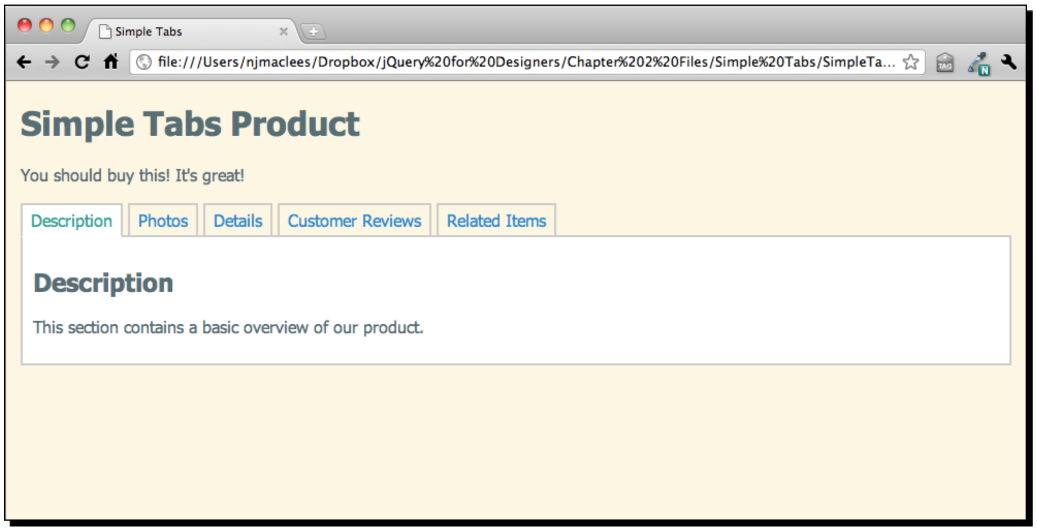
We selected all the links on our page according to the file extension in the `href` attribute and used that to add appropriate class names with jQuery. We then used those class names in our CSS to add icons to each link type with some CSS styles. Site visitors without JavaScript enabled will still be able to click the links and download the associated files. They'll just miss out on the icons that tip them off to the file type behind each link.

Now you can see how jQuery and CSS can work together to add new functionality to your pages. jQuery can modify elements' class names, and CSS can then be used to style those elements according to their class name.

Simple tabs

If we have a large amount of information to present that might not be relevant to all site visitors, we can compress the amount of space the information takes by hiding selected bits of information until the site visitor requests it. One of the most common ways of making all the information available but hidden until requested is tabs. Tabs echo the real-world example of a tabbed notebook or labeled folders in a filing cabinet and are easy for site visitors to understand. And believe it or not, they're also easy to implement with jQuery.

Here's an idea of what our page will look like after we've created our tabs:


Simple Tabs
file:///Users/njmaclees/Dropbox/jQuery%20for%20Designers/Chapter%202%20Files/Simple%20Tabs/SimpleTa...
Description
Photos
Details
Customer Reviews
Related Items

Simple Tabs Product

You should buy this! It's great!

Description

This section contains a basic overview of our product.

Time for action – creating simple tabs

1. We'll get started with our basic HTML file and associated folders, like we created in *Chapter 1, Designer, Meet jQuery*. Inside the `<body>` tag, we'll start by setting up our simple example that will work even for users with JavaScript disabled: we'll put a list of anchor links to different areas of the page at the top, then wrap each of our content sections in a `div` with an `id` as follows:

```
<h1>Simple Tabs Product</h1>
<p>You should buy this, it's great!</p>

<ul>
   <li><a href="#description">Description</a></li>
   <li><a href="#photos">Photos</a></li>
   <li><a href="#details">Details</a></li>
   <li><a href="#reviews">Customer Reviews</a></li>
   <li><a href="#related">Related Items</a></li>
</ul>

<div id="description">
   <h2>Overview</h2>
   <p>This section contains a basic overview of our product.</p>
</div>

<div id="photos">
   <h2>Photos</h2>
   <p>This section contains additional photos of our product.</p>
</div>

<div id="details">
   <h2>Details</h2>
   <p>This is where we list out all the details of our product –
size, weight, color, materials, etc.</p>
</div>

<div id="reviews">
   <h2>Customer Reviews</h2>
   <p>Here's where we would list all of the glowing reviews our
customers had written</p>
</div>

<div id="related">
   <h2>Related Items</h2>
   <p>And here we would list out other super items that our
customers might also like to buy.</p>
</div>
```

If we view this HTML in a browser, we'll see a list of links at the top of the page that when clicked, jump down the page to the appropriate section so that the site visitor can easily find each section without scrolling on their own. We've basically created a clickable table of contents for our page.

2. Now we want to enhance this for our site visitors that have JavaScript enabled. We'll start by adding an `id` to the `<ul>` that contains our table of contents and we'll add a class name to each of the `<div>`s that contain our sections of content—this will make it easier for us to select just the pieces of the page we want with jQuery and will also make it easier for us to style our tabs with CSS.

```
<ul id="tabs">
<li><a href="#description">Description</a></li>
<li><a href="#photos">Photos</a></li>
<li><a href="#details">Details</a></li>
<li><a href="#reviews">Customer Reviews</a></li>
<li><a href="#related">Related Items</a></li>
```

```
</ul>

    <div id="description" class="tab-section">
   <h2>Overview</h2>
   <p>This section contains a basic overview of our product.</p>
</div>
   <div id="photos" class="tab-section">
   <h2>Photos</h2>
   <p>This section contains additional photos of our product.</p>
</div>

   <div id="details" class="tab-section">
   <h2>Details</h2>
   <p>This is where we list out all the details of our product –
size, weight, color, materials, etc.</p>
</div>

   <div id="reviews" class="tab-section">
   <h2>Customer Reviews</h2>
   <p>Here's where we would list all of the glowing reviews our
customers had written</p>
</div>

   <div id="related" class="tab-section">
   <h2>Related Items</h2>
   <p>And here we would list out other super items that our
customers might also like to buy.</p>
</div>
```

3. Next, we'll use jQuery to hide all of our `tab-sections`. Open up the `scripts.js` file inside your `scripts` folder and inside the document `ready` statement, select the `tab-sections` and hide them:

```
$(document).ready(function(){
     $('.tab-section').hide();
});
```

Now when we load the page, we'll only see our table of contents.

4. Next, we need to show the appropriate section when one of our tabs is clicked. We'll start by binding a function to the click event of the links inside our table of contents—just like we did when we opened a link in a new window:

```
$(document).ready(function(){
   $('.tab-section').hide();
```

```
        $('#tabs a').bind('click', function(e){
            e.preventDefault;
        });
});
```

With this bit of code, we've selected all of the links inside the `<ul>` with the `id` of `#tabs` and bound a function to the links on click. So far, all this function does is cancel the click—if you load the page in a browser at this point, you'll see that clicking on the links does nothing—the page no longer jumps down to the associated section.

5. Next, we want to select the appropriate section and show it. To do that, we'll use the hash—or the part of the `href` attribute that includes the `#` symbol.

```
$('#tabs a').bind('click', function(e){
    $(this.hash).show();
    e.preventDefault;
});
```

When I pass `this.hash` to the jQuery function, the `this` I'm dealing with is the link that was just clicked and `this.hash` is the value of the `href` attribute starting with the `#` symbol and continuing to the end. If I were to click on the overview tab, for example, passing `this.hash` to the jQuery function is the same as writing the following:

```
$('#overview')
```

But of course, this is done in a much more flexible way—it will work for any tab linked to any section of the page. So, for example, if I wanted to replace the customer reviews tab with a shipping information tab, I wouldn't have to update my JavaScript, only the HTML markup itself—the JavaScript is flexible enough to adjust to changes.

6. So now when I click on one of the table of contents links, it will show me the associated section, but if I keep clicking on links, the sections just keep showing up, and after clicking all the links, all the sections are visible—that's not what we want. We'll have to hide the visible section and show only the section we want. Let's add a line to our code to select the visible `tab-section` and hide it before we show the new section:

```
$('#tabs a').bind('click', function(e){
        $('.tab-section:visible').hide();
    $(this.hash).show();
    e.preventDefault;
});
```

You're probably familiar with **pseudoclass** selectors in CSS—they're often used to select the hover, visited, and active states of links (`a:hover`, `a:visited`, and `a:active`). jQuery makes a few additional `pseudoclass` selectors available to us—there are pseudoclass selectors for buttons, empty elements, disabled form fields, checkboxes, and more. You can check out all the available selectors for jQuery in the jQuery documentation at `http://api.jquery.com/category/selectors/`. Here, we're using the `:visible` pseudoclass to select the `.tab-section` that's currently visible. Once we've selected the visible `.tab-section`, we hide it and then find the correct `tab-section` and show it.

7. All we need now is some CSS to get our tabs styles to look like a tabbed section of content. Open the `styles.css` file that's inside your `styles` folder, and add some CSS styles as follows. Feel free to customize them to suit your own taste.

```
#tabs   {
   overflow: hidden;
   zoom: 1;
   }

#tabs li   {
   display: block;
   list-style: none;
   margin: 0;
   padding: 0;
   float: left;
   }

#tabs li a   {
   display: block;
   padding: 2px 5px;
   border: 2px solid #ccc;
   border-bottom: 0 none;
   text-align: center;
   }

.tab-section   {
   padding: 10px;
   border: 2px solid #ccc;
   }
```

8. Now if you load this up in a browser, you'll see that there's a little something missing—we should highlight the currently selected tab to make it obvious which one is selected. We can do that by adding a CSS class to the current tab. Go back to your `scripts.js` file and add a bit of code to add a class to the current tab and remove the class from any non-current tabs as follows:

```
$('#tabs a').bind('click', function(e){
      $('#tabs a.current').removeClass('current');
   $('.tab-section:visible').hide();
   $(this.hash).show();
      $(this).addClass('current');
   e.preventDefault;
});
```

First, we'll find the tab that has the class `current`, and remove that class. Then we'll get the tab that was just clicked and add the `current` class to it. That way, we make sure that only one tab will be marked as the current tab at any given time.

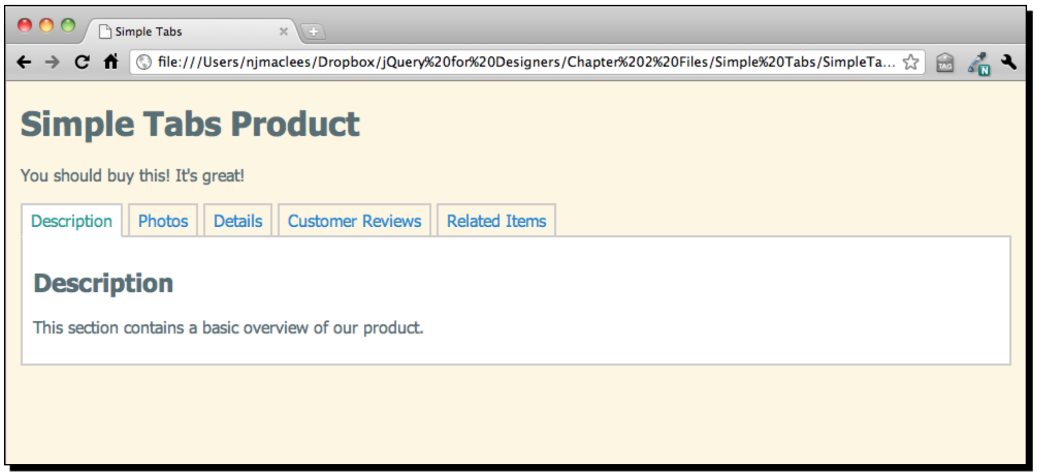
9. Next, we'll add some styles in our CSS for our new class. Open up `styles.css` and add a bit of code to distinguish the currently selected tab. Again, feel free to customize this style to suit your own tastes:

```
#tabs li a.current   {
   background: #fff;
   color: #000;
   }
```

10. So now our tabs are working the way we expect, and the only thing left to do is to make the first tab active and show the first content section when the page is first loaded instead of leaving them all hidden. We've already written the function to do this, so now all we have to do is call it for our first tab:

```
$('#tabs a').bind('click', function(e){
   $('#tabs a.current').removeClass('current');
   $('.tab-section:visible').hide();
   $(this.hash).show();
   $(this).addClass('current');
   e.preventDefault;
   }).filter(':first').click();
```

The jQuery object's `filter()` method will allow us to filter a previously selected set of elements—in this case we're dealing with all of the `<a>` tags inside the `<ul>` with the `id #tabs`. We bind a click function to all of those links, then we'll filter out just the first link using the `:first` pseudoclass made available to us in jQuery and tell jQuery to click the first tab for us—this will run our function, adding the `current` class to the first link, and showing the first `.tab-section`—just the way we would expect the page to look when we load it.


Simple Tabs
file:///Users/njmaclees/Dropbox/jQuery%20for%20Designers/Chapter%202%20Files/Simple%20Tabs/SimpleTa...
Description
Photos
Details
Customer Reviews
Related Items

Simple Tabs Product

You should buy this! It's great!

Description

This section contains a basic overview of our product.

What just happened?

We set up a set of simple tabs with jQuery. For site visitors with JavaScript disabled, the tabs will function like a table of contents at the top of the document, jumping them down to the various sections of content when they're clicked. For site visitors with JavaScript, though, the sections of content will be completely hidden until needed. Clicking on each tab reveals the content associated with that tab. This is a great way to save space in a UI—making all the content available on demand in a small space.

We hid the tab contents with JavaScript instead of with CSS to be sure that users without JavaScript enabled would still be able to access all of our content.

Summary

In this chapter, you learned how to take basic links—the backbone of the Internet—and enhance them to add some new behaviors and capabilities. You learned how to make a link open in a new window, how to add icons to links depending on which type of file was linked to and how to set up a basic tabbed interface. These are all very common requirements for websites and these will serve as great building blocks for you as you learn more about jQuery and JavaScript.

3
Making a Better FAQ Page

The Frequently Asked Questions page has been a mainstay of all types of websites since the dawn of the Web. It's used as a marketing page, as an attempt to reduce the number of calls or e-mails to a customer service department, and as a helpful tool for site visitors to learn more about the company or organization they're dealing with or the products or services they're interested in purchasing.

Though we'll be building an FAQ page for this example, this expand and collapse technique will be useful in many different situations—a list of events with event details, a listing of staff or members with bios, a list of products with details—any situation where a listing of items should be quick and easy for site visitors to scan, but where more information should be readily and easily available upon demand when they find the thing they're looking for.

In this chapter, we'll learn:

- How to traverse an HTML document with jQuery
- How to show and hide elements
- How to use simple jQuery animations
- How to easily toggle a class name for an element

FAQ page markup

We'll get started by taking some extra care and attention with the way we mark up our FAQ list. As with most things dealing with web development, there's no one right way of doing anything, so don't take this approach as the only correct one. Any markup that makes sense semantically and makes it easy to enhance your list with CSS and JavaScript is perfectly acceptable.

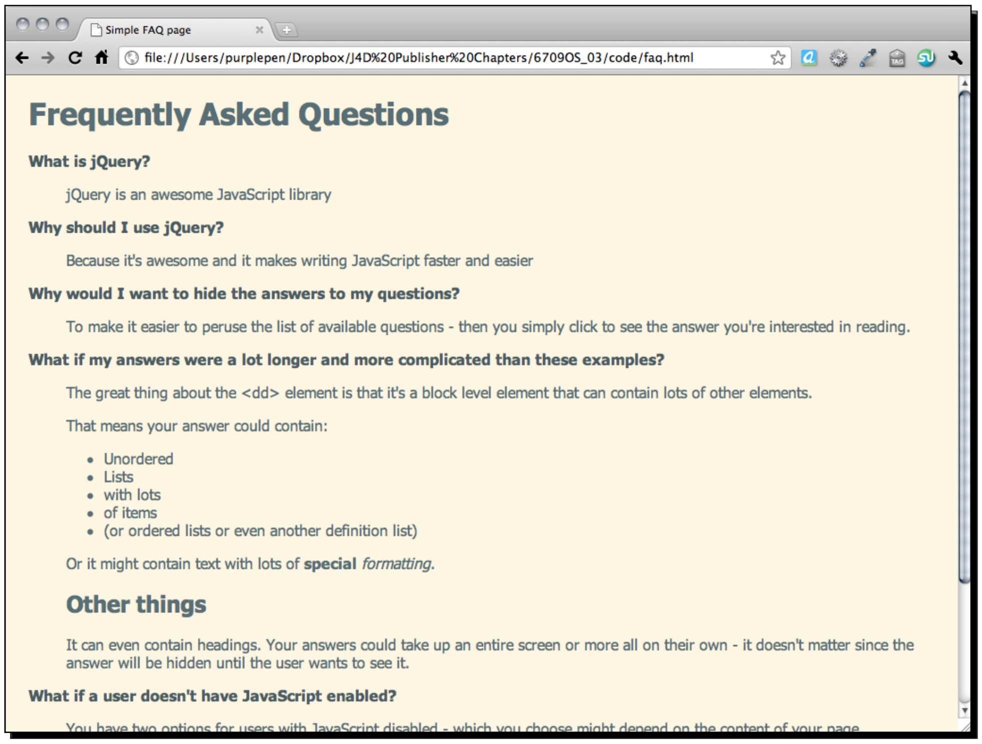
Time for action – setting up the HTML

1. We'll get started with our sample HTML file and associated files and folders, like we set up in *Chapter 1*, *Designer, Meet jQuery*. In this case, our HTML is going to be a definition list with the questions inside the `<dt>` tags and the answers wrapped in `<dd>` tags. By default, most browsers will indent the `<dd>` tags which means the questions hang into the left margin, making them easy to scan. Inside the `<body>` of your HTML document, add a heading and a definition list as follows:

```
<h1>Frequently Asked Questions</h1>
<dl>
       <dt>What is jQuery?</dt>
       <dd>
              <p>jQuery is an awesome JavaScript library</p>
       </dd>

       <dt>Why should I use jQuery?</dt><dd>
              <p>Because it's awesome and it makes writing
JavaScript faster and easier</p>
       </dd>

       <dt>Why would I want to hide the answers to my questions?
</dt>
       <dd>
              <p>To make it easier to peruse the list of available
questions - then you simply click to see the answer you're
interested in reading.</p>
       </dd>

       <dt>What if my answers were a lot longer and more
complicated than these examples?</dt>
       <dd>
```

```
            <p>The great thing about the &lt;dd&gt; element is
that it's a block level element that can contain lots of other
elements.</p>
            <p>That means your answer could contain:</p>
            <ul>
                  <li>Unordered</li>
                  <li>Lists</li>
                  <li>with lots</li>
                  <li>of items</li>
                  <li>(or ordered lists or even another
definition list)</li>
            </ul>
            <p>Or it might contain text with lots of
<strong>special</strong> <em>formatting</em>.</p>
            <h2>Other things</h2>
            <p>It can even contain headings. Your answers could
take up an entire screen or more all on their own - it doesn't
matter since the answer will be hidden until the user wants to see
it.</p>
      </dd>

      <dt>What if a user doesn't have JavaScript enabled?</dt>
      <dd>
            <p>You have two options for users with JavaScript
disabled - which you choose might depend on the content of your
page.</p>
            <p>You might just leave the page as it is - and
make sure the &lt;dt&gt; tags are styled in a way that makes them
stand out and easy to pick up when you're scanning down through
the page. This would be a great solution if your answers are
relatively short.</p>
            <p>If your FAQ page has long answers, it might be
helpful to put a table of contents list of links to individual
questions at the top of the page so users can click it to jump
directly to the question and answer they're interested in. This is
similar to what we did in the tabbed example, but in this case,
we would use jQuery to hide the table of contents when the page
loaded since users with JavaScript wouldn't need to see the table
of contents.</p>
      </dd>
</dl>
```

2. You can adjust the style of the page however you would like by adding in some CSS. Here's how I've styled mine:


Simple FAQ page
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/6709OS_03/code/faq.html

Frequently Asked Questions

What is jQuery?

jQuery is an awesome JavaScript library

Why should I use jQuery?

Because it's awesome and it makes writing JavaScript faster and easier

Why would I want to hide the answers to my questions?

To make it easier to peruse the list of available questions - then you simply click to see the answer you're interested in reading.

What if my answers were a lot longer and more complicated than these examples?

The great thing about the <dd> element is that it's a block level element that can contain lots of other elements.

That means your answer could contain:

- Unordered
- Lists
- with lots
- of items
- (or ordered lists or even another definition list)

Or it might contain text with lots of **special** *formatting*.

Other things

It can even contain headings. Your answers could take up an entire screen or more all on their own - it doesn't matter since the answer will be hidden until the user wants to see it.

What if a user doesn't have JavaScript enabled?

You have two options for users with JavaScript disabled - which you choose might depend on the content of your page

For users with JavaScript disabled, this page works fine as is. The questions hang into the left margin and are bolder and darker than the rest of the text on the page, making them easy to scan.

What just happened?

We set up a basic definition list to hold our questions and answers. The default style of the definition list lends itself nicely to making the list of questions scannable for site visitors without JavaScript. We can enhance that further with our own custom CSS to make the style of our list match our site.

Time for action – moving around an HTML document

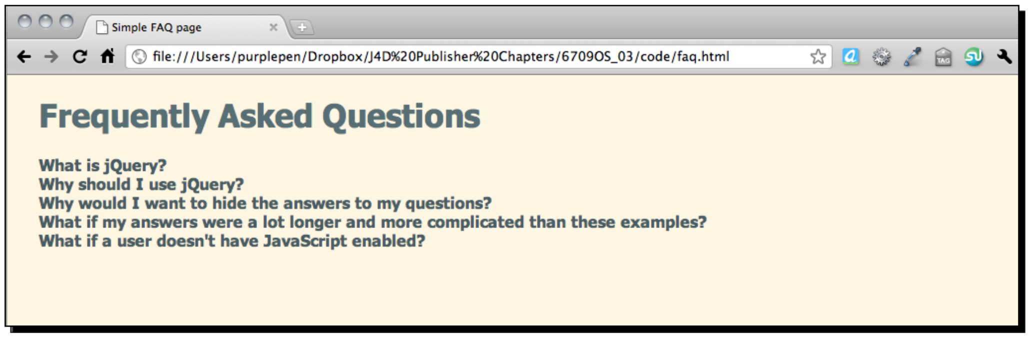
1. We're going to keep working with the files we set up in the previous section. Open up the `scripts.js` file that's inside your `scripts` folder. After the document ready statement, write a new empty function called `dynamicFaq`:

```
function dynamicFaq() {
	//our FAQ code will go here
}
```

2. Let's think through how we would like this page to behave. We would like to have all the answers to our questions hidden when the page is loaded, then when a user finds the question they're looking for, we would like to show the associated answer when they click the question.

   That means the first thing we'll need to do is hide all the answers when the page loads. That's as simple as selecting all of our `<dd>` elements and hiding them. Inside your `dynamicFaq` function, add a line of code to hide the `<dd>` elements:

```
function dynamicFaq() {
   $('dd').hide();
}
```


Simple FAQ page
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/67090S_03/code/faq.html

Frequently Asked Questions

What is jQuery?
Why should I use jQuery?
Why would I want to hide the answers to my questions?
What if my answers were a lot longer and more complicated than these examples?
What if a user doesn't have JavaScript enabled?

You might be wondering why we didn't use CSS to set the display of the `<dd>` tags to `none`. That would have hidden our answers, but it would have hidden our answers for everyone—site visitors without JavaScript enabled wouldn't have been able to access the answers, the most important part of our page!

It would also stop most search engines from indexing the content inside of our answers, which could be helpful for people trying to find the answers to their questions in a search engine. By using JavaScript to hide the answers, we can be sure the answers will be available unless the user has JavaScript enabled and is capable of showing them again.

3. Now, we need to show the answer when the site visitor clicks on a question. To do that, we need to tell jQuery to do something whenever someone clicks on one of the questions or `<dt>` tags. Inside the `dynamicFaq` function, add a line of code to bind a click function to the `<dt>` tags:

```
function dynamicFaq() {
   $('dd').hide();
   $('dt').bind('click', function(){
          //Show function will go here
   });
}
```

4. When the site visitor clicks on a question, we want to get the answer to that question and show it because our FAQ list is set up similar to the following code:

```
<dl>
   <dt>Question 1</dt>
   <dd>Answer to Question 1</dd>
   <dt>Question 2</dt>
   <dd>Answer to Question 2</dd>
   …
</dl>
```

...we know that the answer is the next node or element in the DOM after our question. We'll start from the question. When a site visitor clicks a question, we can get the current question by using jQuery's `$(this)` selector. The user has just clicked on a question, and we say `$(this)` to mean the question they just clicked on. Inside that new `click` function, add `$(this)` so we can refer to the clicked question:

```
$('dt').bind('click', function(){
   $(this);
});
```

5. Now that we have the question that was just clicked, we need to get the next thing or the answer to that question so that we can show it. This is called **traversing the DOM** in JavaScript speak. It just means that we're moving to a different part of the document.

jQuery gives us the `next()` method for moving to the next node in the DOM. We'll select our answer by doing as follows:

```
$('dt').bind('click', function(){
   $(this).next();
});
```

6. Now we've moved from the question to the answer. Now all that's left to do is show the answer:

```
$('dt').bind('click', function(){
   $(this).next().show();
});
```

7. Don't forget that our `dynamicFaq` function won't do anything until we call it. Call the `dynamicFaq` function inside your document ready statement:

```
$(document).ready(function(){
  dynamicFaq();
});
```

8. Now, if we load the page in the browser, you can see that all of our answers are hidden until we click on the question. This is nice and useful, but it would be even nicer if the site visitor could hide the answer again when they're done reading it to get it out of their way. Luckily, this is such a common task, jQuery makes this very easy for us. All we have to do is replace our call to the `.show()` method with a call to the `.toggle()` method as follows:

```
$('dt').bind('click', function(){
   $(this).next().toggle();
});
```

Now when you refresh the page in the browser, you'll see that clicking the question once shows the answer and clicking the question a second time hides the answer again.

What just happened?

Toggling the display of elements on a page is a common JavaScript task, so jQuery already has built-in methods for handling it and making it simple and straightforward to get this up and running on our page. That was pretty easy; just a few lines of code.

Sprucing up our FAQ page

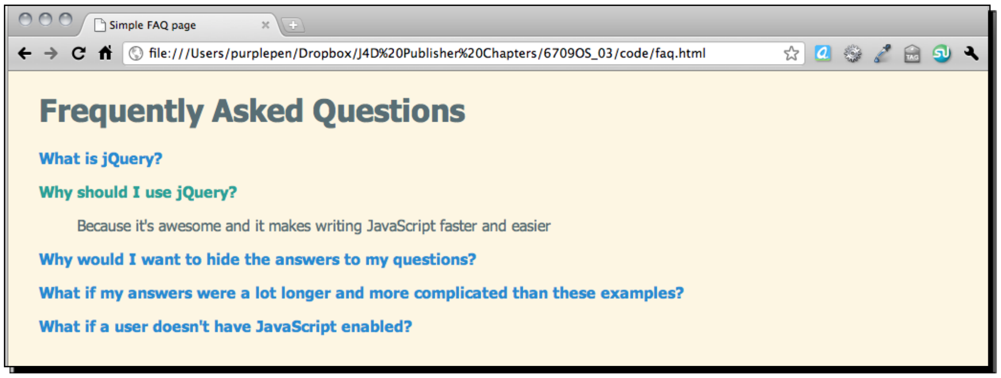
That was so easy, in fact we have plenty of time left over for enhancing our FAQ page to make it even better. This is where the power of jQuery becomes apparent—you can not only create a show/hide FAQ page, but you can make it a fancy one and still meet your deadline. How's that for impressing a client or your boss?

Time for action – making it fancy

1. Let's start with a little CSS to change the cursor to a pointer and add a little hover effect to our questions to make it obvious to site visitors that the questions are clickable. Open up the `styles.css` file that's inside the styles folder and add this bit of CSS:

```
dt      {
        color: #268bd2;
        font-weight: bold;
        cursor: pointer;
        margin: 0 0 1em 0;
        }

dt:hover      {
        color: #2aa198;
        }
```

   That definitely helps to communicate to the site visitor that the questions are clickable.



Simple FAQ page
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/6709OS_03/code/faq.html

Frequently Asked Questions

What is jQuery?

Why should I use jQuery?

Because it's awesome and it makes writing JavaScript faster and easier

Why would I want to hide the answers to my questions?

What if my answers were a lot longer and more complicated than these examples?

What if a user doesn't have JavaScript enabled?

2. When we click a question to see the answer, the change isn't communicated to the site visitor very well – the jump in the page is a little disconcerting and it takes a moment to realize what just happened. It would be nicer and easier to understand if the question were to slide into view; the site visitor could literally see the question appearing and would understand immediately what change just happened on the screen.

   jQuery makes it easy for us. We just have to replace our call to the `.toggle()` method with a call to the `.slideToggle()` method.

```
$('dt').bind('click', function(){
   $(this).next().slideToggle();
});
```

Now if you view the page in your browser, you can see that the questions slide smoothly into and out of view when the question is clicked. It's easy to understand what's happening when the page changes, and the animation is a nice touch.

What just happened?

We replaced our `toggle()` method with the `slideToggle()` method to animate the showing and hiding of the answers. This makes it easier for the site visitor to understand the change that's taking place on the page. We also added some CSS to make the questions appear to be clickable to communicate the abilities of our page to our site visitors.

We're almost there!

jQuery made animating that show and hide so easy that we've still got time left over to enhance our FAQ page even more. It would be nice to add some sort of indicator to our questions to show that they're collapsed and can be expanded, and to add some sort of special style to our questions once they're opened to show that they can be collapsed again.

Time for action – adding some final touches

1. Let's start with some simple CSS to add a small arrow icon to the left side of our questions. Head back into `style.css` and modify the styles a bit to add an arrow icon or an icon of your choosing. You can place the icon you choose inside your `images` folder:

```
dt     {
       color: #268bd2;
       font-weight: bold;
       cursor: pointer;
       margin: 0 0 1em 0;
       padding: 0 0 0 20px;
       background: url(../images/arrow.png) 0 0 no-repeat;
       line-height: 16px;
       }

dt:hover      {
       color: #2aa198;
       background-position: 0 -32px;
       }
```

I'm using an image sprite for the arrows. As I'm changing the color of my questions from blue to green when the mouse hovers over the questions, I've included both the blue and green arrows in my sprite and am using a bit of CSS to show the green arrow when the text turns green. That means only one image has to download and there's no delay in downloading a new image to show when the mouse hovers over my question. If you're unfamiliar with the CSS image sprite technique, I recommend taking a look at *Chris Coyier's* article explaining it at `http://css-tricks.com/css-sprites/`.

2. Now, we want to change the arrow to a different orientation when the question is opened. All we have to do is use a new CSS class for the open state of our questions and code the off and on states so the new arrow shape changes color as well. Again, I've included these arrow images in the same sprite, so the only thing I have to change is the background position:

```
dt.open      {
       background-position: 0 -64px;
       }

dt.open:hover {
       background-position: 0 -96px;
       }
```

Just make sure to add these new classes *after* the other CSS we're using to style our `<dt>` tags. That will ensure that the CSS cascades the way we intended.

3. So we have our CSS to show our questions are open, but how do we actually get to use it? We'll use jQuery to add the class to our question when it is opened, and to remove the class when it's closed.

jQuery provides some nice methods for working with CSS classes. `addClass()` will add a class to a jQuery object and `removeClass()` will remove a class. However, we want to toggle our class just like we're toggling the show and hide of our questions. jQuery's got us covered for that too. We want the class to change when we click on the question, so we'll add a line of code inside our `dynamicFaq` function that we're calling each time a `<dt>` is clicked:

```
$('dt').bind('click', function(){
   $(this).toggleClass('open');
       $(this).next().slideToggle();
});
```

Now when you view the page, you'll see your open styles being applied to the `<dt>` tags when they're open and removed again when they're closed. But we can actually crunch our code to be a little bit smaller.

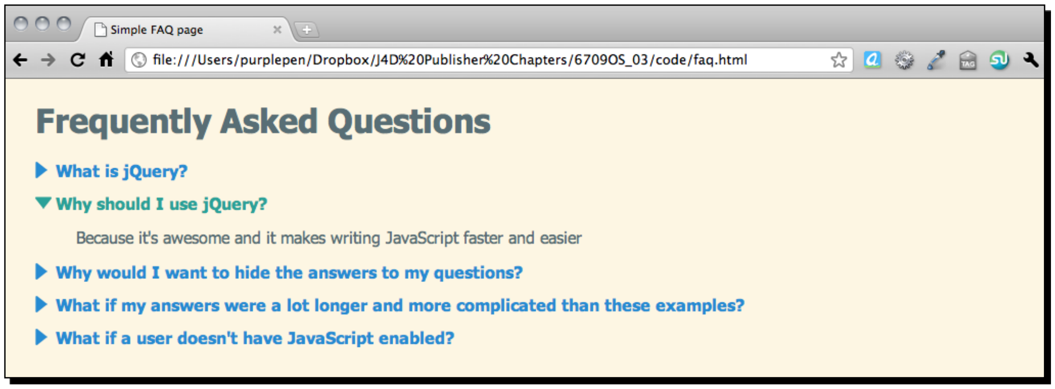
4. One of the most powerful features of jQuery is called chaining. We've already used chaining in our code when we added `slideToggle()` to the `next()` method on one line.

    ```
    $(this).next().slideToggle();
    ```

    Methods in jQuery can be chained. You can keep adding new methods to further transform, modify or animate an element. This line of code gets the question, traverses the DOM to the next node, which we know is our `<dd>`, and then toggles the slide animation for that `<dd>`.

    We can take advantage of chaining again. We have a bit of redundancy in our code because we're starting two different lines with `$(this)`. We can remove that extra `$(this)` and just add our `toggleClass()` method to the chain we've already started as follows:

    ```
    $(this).toggleClass('open').next().slideToggle();
    ```


Simple FAQ page
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/6709OS_03/code/faq.html

Frequently Asked Questions

▶ What is jQuery?

▼ Why should I use jQuery?

Because it's awesome and it makes writing JavaScript faster and easier

▶ Why would I want to hide the answers to my questions?

▶ What if my answers were a lot longer and more complicated than these examples?

▶ What if a user doesn't have JavaScript enabled?

What just happened?

We created CSS styles to style the open and closed states of our questions, and then we added a bit of code to our JavaScript to change the CSS class of the question to use our new styles. jQuery provides a few different methods for updating CSS classes, which is often a quick and easy way to update the display of our document in response to input from the site visitor. In this case, since we wanted to add and remove a class, we used the `toggleClass()` method. That saved us from having to figure out on our own if we needed to add or remove the open class.

We also took advantage of chaining to simply add this new functionality to our existing line of code, making the animated show and hide of the answer and the change of CSS class of our question happen all in just one line of code. How's that for impressive power in a short amount of code?

Summary

In this chapter, we learned how to set up a basic FAQ page that hides the answers to the questions until the site visitor needs to see them. Because jQuery made that so simple, we had plenty of time left over for enhancing our FAQ page even more, adding animations to our show and hide of the questions, and taking advantage of CSS to style our questions with special open and closed classes to communicate to our site visitors how our page works. And we did all of that with just a few lines of code.

Next, we'll learn how to use custom scrollbars on our pages.

4
Building Custom Scrollbars

A common strategy for dealing with pages that have a lot of content is to hide some of the content until the site visitor wants or needs it. There are many approaches to this—you could use tabs, accordions, lightboxes, or the focus of this chapter, scrollable areas.

Scrollable areas are easy for site visitors to understand and use, but they often get ignored because some operating systems have unsightly scrollbars that ruin the aesthetics of your carefully-tuned design. Browsers offer few, if any, options for customizing the appearance of scrollbars, and no official means of doing so has ever been included in any HTML or CSS specification.

Some designers have turned to Flash to create custom scrollbars, and I'm sure you've come across samples of these Flash scrollbars online – more often than not, they're unwieldy and break common conventions for dealing with scrollable areas. For example, you're rarely able to use your mouse's scrollwheel to scroll through a Flash scrollable area.

In this chapter, we'll learn:

- How to download and use jQuery plugins to do even more with jQuery
- How to use a plugin's built-in customization options to customize how a plugin works
- How to use CSS to customize a plugin even further
- How to set up custom-designed scrollbars that work just as your site visitors expect
- How to use the jScrollPane plugin to smoothly scroll between different bits of content in our scrollable area

Designer, meet plugins

We've already talked about how programmers solve the same problems over and over again. It's these common tasks that jQuery simplifies so that we can accomplish these tasks with a minimum amount of code. But what about the tasks that are only somewhat common, like the desire for beautiful custom scrollbars that work?

That's where the jQuery community becomes important. Developers in the jQuery community are able to write code that extends the functionality of jQuery to simplify tasks that are only somewhat common. These bits of code are called **Plugins** and they are used in conjunction with the jQuery library to make coding complex interactions, widgets, and effects as simple as using the features already built into jQuery.

You'll find a library of hundreds of jQuery plugins on the official jQuery site. In addition to those, there are literally thousands more available from sites across the Web for just about any task you want to accomplish.

To create custom scrollbars, we'll be using *Kelvin Luck's* jScrollPane plugin. You'll learn how to install the plugin on your page and how to configure the CSS and options to make your scrollbars look and work the way you want.

Choosing a plugin

Recently, the jQuery team has started supporting a small number of official jQuery plugins, and you can use those confidently, knowing that they have the same level of expertise, documentation, and support behind them that jQuery itself has. All other jQuery plugins are provided by various members of the jQuery community, and those authors are solely responsible for documentation and support for their own plugins. Writing and providing jQuery plugins is a bit of a free-for-all, and sadly you will come across a fair number of jQuery plugins which are poorly documented, poorly supported, and even worse, poorly written. What kinds of things should you, as a newcomer to jQuery, look for when choosing a plugin?

- *A recent update to the plugin*. Frequent updates mean that a plugin is well-supported and that the author is keeping the plugin up to date as jQuery and browsers evolve.
- *Thorough and easy-to-understand documentation*. Before attempting to download and use a plugin, take a look through the plugin's documentation and make sure you understand how to implement the plugin and how to use any options the plugin makes available to you.
- *Browser support*. Great plugins generally have the same browser support as the jQuery library itself.

- *Working demo*. Most plugin authors will offer one or more working demos of their plugin in action. Check out the demo(s) in as many different browsers as possible to be sure the plugin works as advertised.
- *Reviews and ratings*. You won't find reviews and ratings for all plugins, but if you can find some, they can be helpful indicators of the quality and reliability of the plugin.

Setting up some scrollable HTML

Let's take a look at how to set up a simple HTML page that contains a scrollable area. Once we've got that out of the way, we'll look at how to replace the default scrollbars with custom ones.

Time for action – scrollable HTML

Follow these steps to set up a simple HTML page with a scrollable area:

1. We'll start off with setting up a basic HTML page and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*. We need to have an area of content that's large enough to scroll, so we'll add several paragraphs of text to the body of the HTML document:

```
<!DOCTYPE html>
<html>
<head>
       <title>Custom Scrollbars</title>
       <link rel="stylesheet" href="styles/styles.css"/>
</head>
<body>

<h2>We don't want this box of content to get too long, so we'll
make it scroll:</h2>
<p>Lorem ipsum dolor sit amet...
Include several paragraphs of lorem ipsum here
...mollis arcu tincidunt.</p>
<script src="scripts/jquery.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

I have not included it all, but I have included five long paragraphs of lorem ipsum text on my page to add some length and give us something to scroll. In case you're not aware, lorem ipsum is simply dummy filler text. You can generate some random lorem ipsum text for yourself to fill your page at `http://lipsum.com`.

2. Now, we need to make our text scroll. To do that, I'm going to wrap all those paragraphs of lorem ipsum in a `div` and then use CSS to set a height on the `div` and set the `overflow` to `auto`:

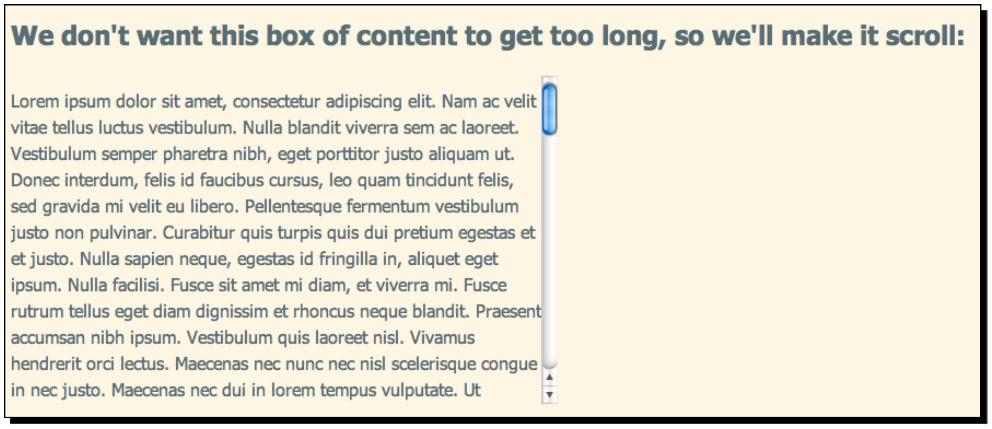
```
<h2>We don't want this box of content to get too long, so we'll
make it scroll:</h2>
<div id="scrolling">
       <p>Lorem ipsum dolor sit amet...
       Include several paragraphs of lorem ipsum here
       ...mollis arcu tincidunt.</p>
</div>
```

3. Next, open your empty `styles.css` file, and add this bit of CSS to make our text area scrollable:

```
#scrolling   {
       width:500px;
       height:300px;
       overflow:auto;
       }
```

Feel free to add some additional CSS to style your text any way you'd like.

Now, when I view my page in a browser, I'll see that the browser has added some (ugly) scrollbars for my text:



We don't want this box of content to get too long, so we'll make it scroll:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam ac velit vitae tellus luctus vestibulum. Nulla blandit viverra sem ac laoreet. Vestibulum semper pharetra nibh, eget porttitor justo aliquam ut. Donec interdum, felis id faucibus cursus, leo quam tincidunt felis, sed gravida mi velit eu libero. Pellentesque fermentum vestibulum justo non pulvinar. Curabitur quis turpis quis dui pretium egestas et et justo. Nulla sapien neque, egestas id fringilla in, aliquet eget ipsum. Nulla facilisi. Fusce sit amet mi diam, et viverra mi. Fusce rutrum tellus eget diam dignissim et rhoncus neque blandit. Praesent accumsan nibh ipsum. Vestibulum quis laoreet nisl. Vivamus hendrerit orci lectus. Maecenas nec nunc nec nisl scelerisque congue in nec justo. Maecenas nec dui in lorem tempus vulputate. Ut

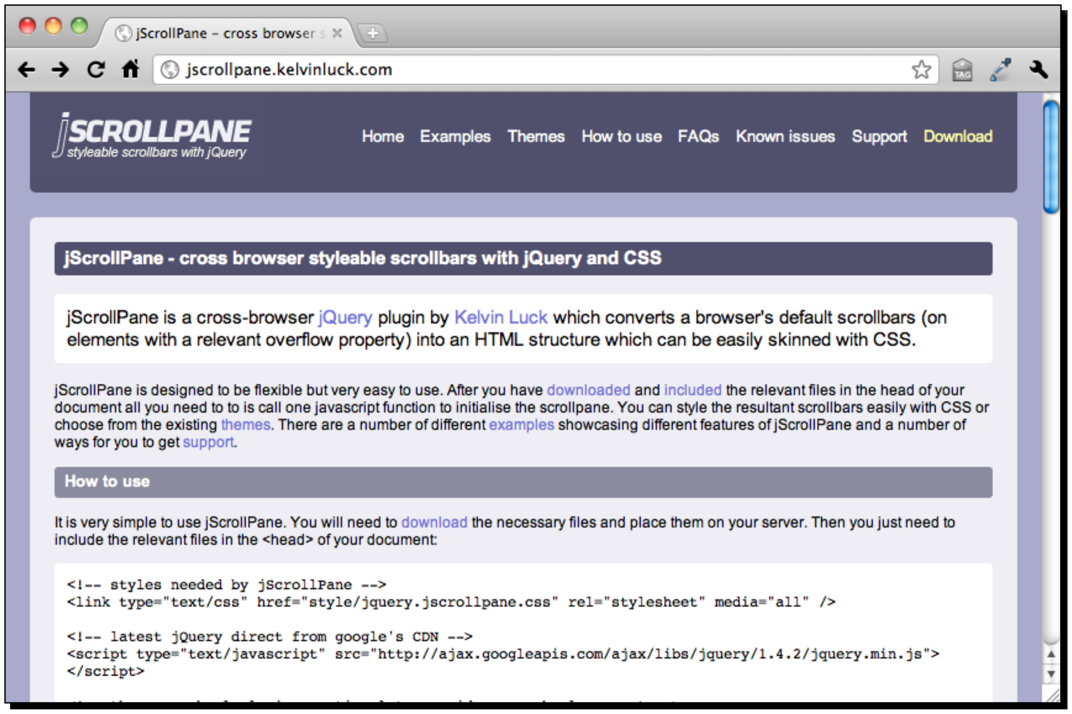
Adding custom scrollbars

In most cases, the appearance of the scrollbars is determined by the operating system your site visitor is using instead of their browser. So it doesn't matter if you're using Firefox, Safari, Chrome, or some other browser on a Mac—you'll always see those trademark shiny blue scrollbars. On a PC, you'll always see chunky squarish scrollbars in whatever color scheme you've set in your Windows options.

Time for action – simple custom scrollbars

You can see that the operating system's default scrollbars stick out like a sore thumb in the middle of our nicely designed page. Let's fix that, shall we?

1. First, we've got to get our hands on the plugin we'd like to use to create our custom scrollbars. Head over to `http://jscrollpane.kelvinluck.com/` and click on the **Download** link in the navigation menu:


jScrollPane – cross browser
jscrollpane.kelvinluck.com
JSCROLLPANE
styleable scrollbars with jQuery
Home
Examples
Themes
How to use
FAQs
Known issues
Support
Download

jScrollPane - cross browser styleable scrollbars with jQuery and CSS

jScrollPane is a cross-browser jQuery plugin by Kelvin Luck which converts a browser's default scrollbars (on elements with a relevant overflow property) into an HTML structure which can be easily skinned with CSS.

jScrollPane is designed to be flexible but very easy to use. After you have downloaded and included the relevant files in the head of your document all you need to to is call one javascript function to initialise the scrollpane. You can style the resultant scrollbars easily with CSS or choose from the existing themes. There are a number of different examples showcasing different features of jScrollPane and a number of ways for you to get support.

How to use

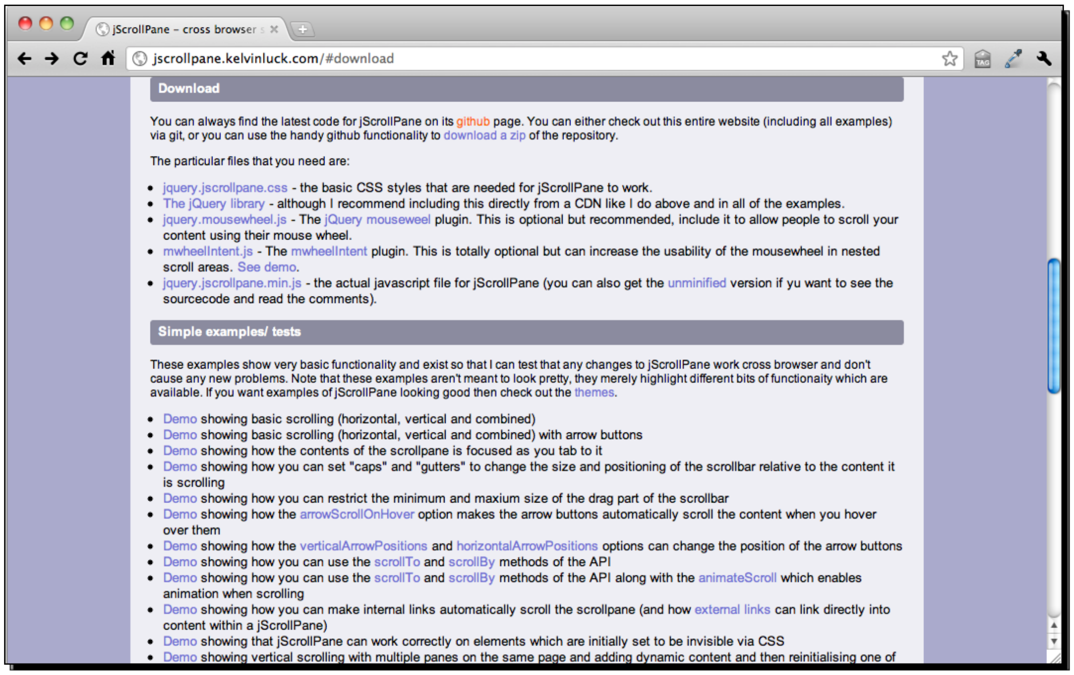
It is very simple to use jScrollPane. You will need to download the necessary files and place them on your server. Then you just need to include the relevant files in the <head> of your document:

```
<!-- styles needed by jScrollPane -->
<link type="text/css" href="style/jquery.jscrollpane.css" rel="stylesheet" media="all" />

<!-- latest jQuery direct from google's CDN -->
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
```

This will jump you down to the **Download** section of the site, where you'll see *Kelvin Luck* is using Github to host his code. Github is a social coding hub – a sort of Facebook for developers – where the main focus is on writing, sharing, and discussing code. Hosting jQuery plugins and other open source code projects with Github is becoming more and more common these days as Github offers developers an easy way to share and collaborate on their code with others.

Don't worry – downloading a plugin from Github is simple. I'll walk you through it.

2. First, click the Github link on *Kelvin Luck's* site:


jScrollPane – cross browser s
jscrollpane.kelvinluck.com/#download

Download

You can always find the latest code for jScrollPane on its github page. You can either check out this entire website (including all examples) via git, or you can use the handy github functionality to download a zip of the repository.

The particular files that you need are:

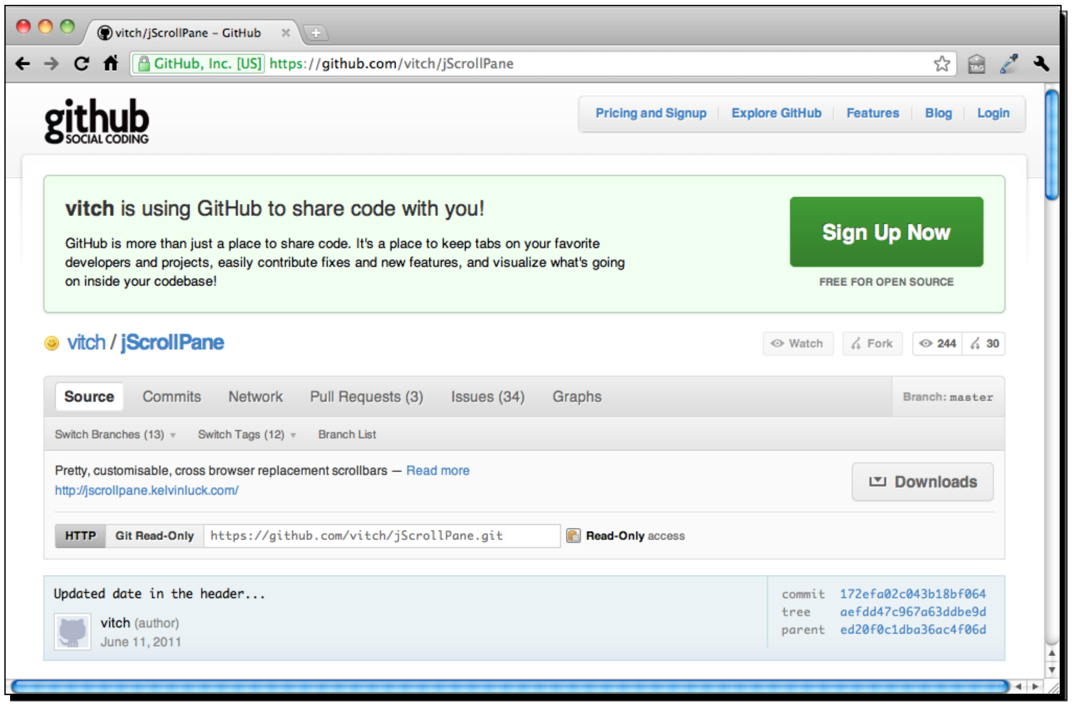
- jquery.jscrollpane.css - the basic CSS styles that are needed for jScrollPane to work.
- The jQuery library - although I recommend including this directly from a CDN like I do above and in all of the examples.
- jquery.mousewheel.js - The jQuery mouseweel plugin. This is optional but recommended, include it to allow people to scroll your content using their mouse wheel.
- mwheelIntent.js - The mwheelIntent plugin. This is totally optional but can increase the usability of the mousewheel in nested scroll areas. See demo.
- jquery.jscrollpane.min.js - the actual javascript file for jScrollPane (you can also get the unminified version if yu want to see the sourcecode and read the comments).

Simple examples/ tests

These examples show very basic functionality and exist so that I can test that any changes to jScrollPane work cross browser and don't cause any new problems. Note that these examples aren't meant to look pretty, they merely highlight different bits of functionaity which are available. If you want examples of jScrollPane looking good then check out the themes.

- Demo showing basic scrolling (horizontal, vertical and combined)
- Demo showing basic scrolling (horizontal, vertical and combined) with arrow buttons
- Demo showing how the contents of the scrollpane is focused as you tab to it
- Demo showing how you can set "caps" and "gutters" to change the size and positioning of the scrollbar relative to the content it is scrolling
- Demo showing how you can restrict the minimum and maxium size of the drag part of the scrollbar
- Demo showing how the arrowScrollOnHover option makes the arrow buttons automatically scroll the content when you hover over them
- Demo showing how the verticalArrowPositions and horizontalArrowPositions options can change the position of the arrow buttons
- Demo showing how you can use the scrollTo and scrollBy methods of the API
- Demo showing how you can use the scrollTo and scrollBy methods of the API along with the animateScroll which enables animation when scrolling
- Demo showing how you can make internal links automatically scroll the scrollpane (and how external links can link directly into content within a jScrollPane)
- Demo showing that jScrollPane can work correctly on elements which are initially set to be invisible via CSS
- Demo showing vertical scrolling with multiple panes on the same page and adding dynamic content and then reinitialising one of

3. That will take you to the jScrollPane project's home page on Github. On the right side of the page, you'll see a **Downloads** button:



vitch is using GitHub to share code with you!

GitHub is more than just a place to share code. It's a place to keep tabs on your favorite developers and projects, easily contribute fixes and new features, and visualize what's going on inside your codebase!

Pretty, customisable, cross browser replacement scrollbars — Read more
http://jscrollpane.kelvinluck.com/

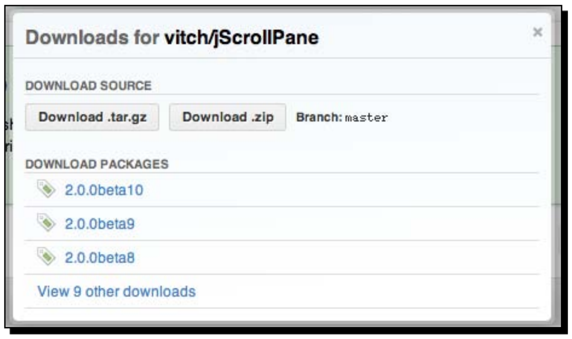
```
https://github.com/vitch/jScrollPane.git
```

```
Updated date in the header...
```

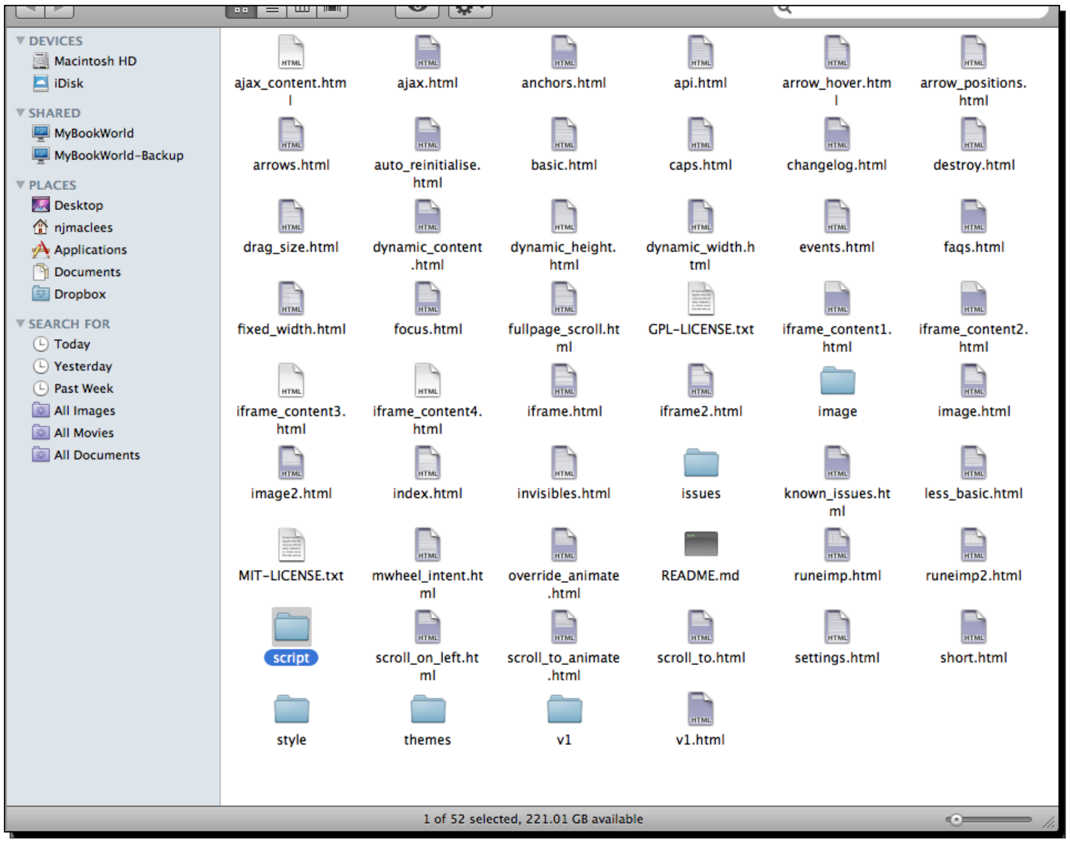
vitch (author)
June 11, 2011

```
commit  172efa02c043b18bf064
tree    aefdd47c967a63ddbe9d
parent  ed20f0c1dba36ac4f06d
```

4. After you click on the **Downloads** button, you'll get a modal dialog window showing all the available download packages for the project. Keep it simple, just click on the **Download .zip** button to get the latest version:

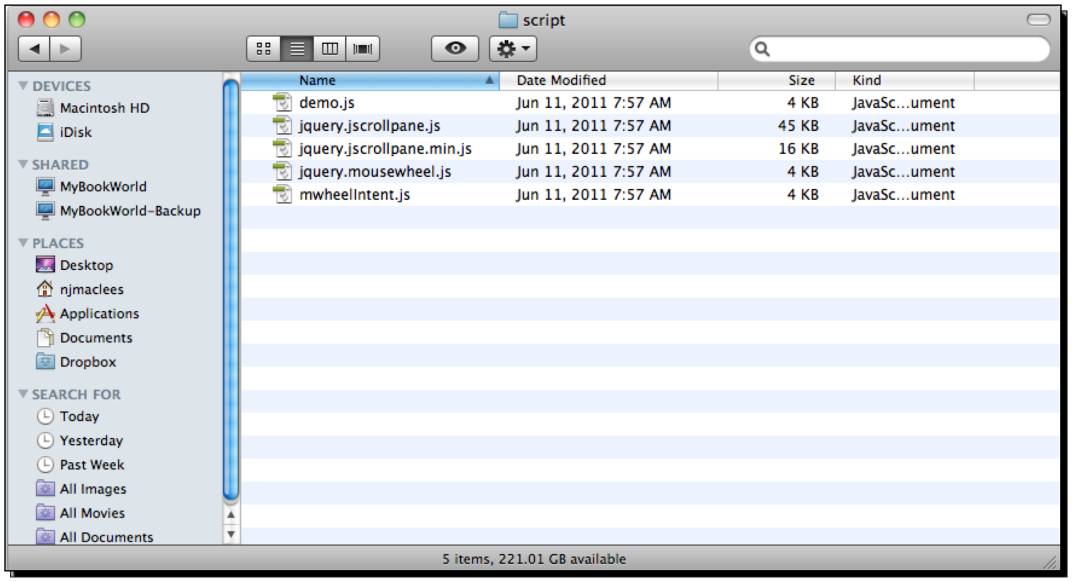


5. The ZIP download will kick off automatically. Once it's done, we're done at Github. I told you it was easy. Now, let's unzip the package and see what's inside.



Wow! That's a lot of files! What are we supposed to do with all of these?

It looks a little scary and confusing, but most of these files are examples and documentation about how to use the plugin. All we need to do is find the JavaScript files that make up the plugin. We'll find those inside the `script` folder.



Name	Date Modified	Size	Kind
demo.js	Jun 11, 2011 7:57 AM	4 KB	JavaSc...ument
jquery.jscrollpane.js	Jun 11, 2011 7:57 AM	45 KB	JavaSc...ument
jquery.jscrollpane.min.js	Jun 11, 2011 7:57 AM	16 KB	JavaSc...ument
jquery.mousewheel.js	Jun 11, 2011 7:57 AM	4 KB	JavaSc...ument
mwheelIntent.js	Jun 11, 2011 7:57 AM	4 KB	JavaSc...ument

5 items, 221.01 GB available

6. Inside the `script` folder, we'll find more like what we expected. Let's figure out what these files are.

- `demo.js` is sample code. It's what *Kelvin Luck* used to put together the assorted demos in the zip file. It might be useful to look at for examples if we get stuck, but we don't need it for our own project.
- `jquery.jscrollpane.js` is the source code for the jScrollPane plugin. If we wanted to modify the way the plugin works or dig through the source code, we could use this file, but we're not expert coders just yet, so we can leave this one alone for now. Why does the filename begin with `jquery.`? It's a common practice to add the `jquery.` in front of the file name to mark it as a jQuery plugin. It can make finding the jQuery plugins much easier in large projects that could be using a dozen or more jQuery plugins along with other JavaScript files.
- `jquery.jscrollpane.min.js` is the compressed version of the plugin. It's the same code as `jquery.jscrollpane.js` except it's been minified. That just means all the extra spaces, tabs, and so on have been removed to make the file smaller – and you can see that it was pretty effective. The minified file is only 16 KB as opposed to 45 KB for the regular file. We won't be able to read this file easily if we open it, but that's fine. We don't need to be able to read it, and it's more important that we serve up the smallest files possible to our site visitors.

- `jquery.mousewheel.js` is the other plugin that we'll be using for our custom scrollbars. It's the plugin that will let our mouse's scrollwheel work just as it should in our scrollable areas
- `mwheelintent.js` is yet another plugin. Looking through *Kelvin Luck's* documentation, we see that this plugin is used for making sure our scrollable areas work as we expect when we nest scrollable areas inside one another. We won't be needing that for now.

7. Copy `jquery.jscrollpane.min.js` and `jquery.mousewheel.js` and put them in your `scripts` folder inside your own project, right next to the `jquery.js` file.

8. Next, we need to include these two files in our page, just like we did with jQuery. Go down to the bottom of your page, and attach the new files between the jQuery `<script>` tag and your own `<script>` tag:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.mousewheel.js"></script>
<script src="scripts/jquery.jscrollpane.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

Anytime you are using jQuery plugins, you want to make sure that you put your `<script>` tags in the correct order. The jQuery `<script>` tag should always be first, any plugins will come next. And finally any script that you're writing that's using jQuery or the plugins will come last. This is because the browser will load these scripts in the order we specify. The plugin files need to be loaded after jQuery because they are using the jQuery library and rely on it being available before they can work. In JavaScript-speak, we call this a dependency. The plugin code is dependent on jQuery. And in turn, our own code is dependent on both the plugin code and the jQuery library itself, so it needs to be loaded after those are available.

In this case, we have an additional dependency that we need to be aware of. The jScrollPane plugin is dependent on the MouseWheel plugin. For this reason, we need to make sure that we load up the MouseWheel plugin first, and then the jScrollPane plugin. If you ever have problems getting jQuery or a plugin to work, it's a good idea to check your script order – a missing or out-of-order dependency is often to blame.

We're almost ready to get our scrollbars set up, but there's one more file that we'll need to include. The jScrollPane plugin actually works by hiding the browser's native scrollbars and constructing replacements from ordinary `<div>`s and `<span>`s. That means we'll need some CSS to style those `<div>`s and `<span>`s to look like a scrollbar. Later on, we'll look at how we can write our own CSS to make our scrollbars look any way we want, but for now, we'll use the CSS that Kelvin Luck has supplied with his plugin to keep things simple.

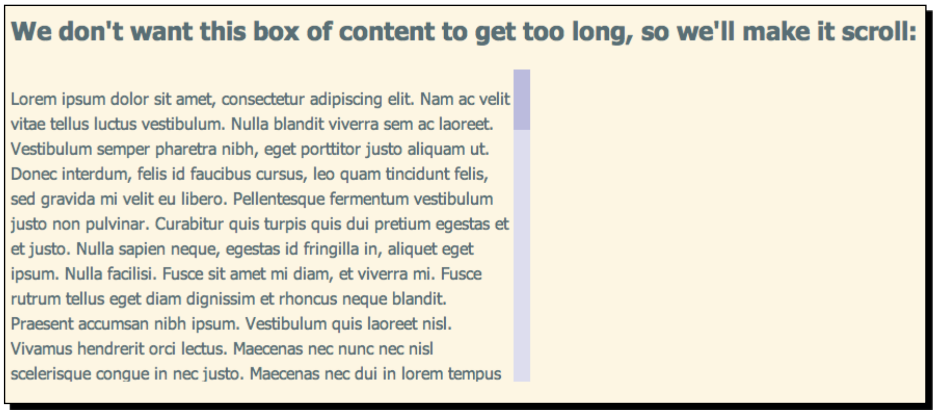
9. Go back into the files we downloaded from Github and find the `style` folder. Inside the folder, you'll find two files: `demo.css` and `jquery.jscrollpane.css`. Just like with the script files, the `demo.css` file is special code that was written just for the examples, but `jquery.jscrollpane.css` is the file that will style our scrollbars. Copy that file to your own `styles` folder and then inside the `<head>` section of your document, attach the new stylesheet before your own `styles.css` file:

```
<head>
       <title>Custom Scrollbars</title>
       <link rel="stylesheet" href="styles/jquery.jscrollpane.css"/>
       <link rel="stylesheet" href="styles/styles.css"/>
</head>
```

10. Phew! We've done a lot of work already, but we still need to add our custom scrollbars to our page. No worries, in true jQuery style, it's just a couple of lines of code. Open up your `scripts.js` file and add this bit of code:

```
$(document).ready(function(){
       $('#scrolling').jScrollPane();
});
```

Now, if you refresh the page, you'll see our scrollable area now has a jScrollPane-style scrollbar.



We don't want this box of content to get too long, so we'll make it scroll:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam ac velit vitae tellus luctus vestibulum. Nulla blandit viverra sem ac laoreet. Vestibulum semper pharetra nibh, eget porttitor justo aliquam ut. Donec interdum, felis id faucibus cursus, leo quam tincidunt felis, sed gravida mi velit eu libero. Pellentesque fermentum vestibulum justo non pulvinar. Curabitur quis turpis quis dui pretium egestas et et justo. Nulla sapien neque, egestas id fringilla in, aliquet eget ipsum. Nulla facilisi. Fusce sit amet mi diam, et viverra mi. Fusce rutrum tellus eget diam dignissim et rhoncus neque blandit. Praesent accumsan nibh ipsum. Vestibulum quis laoreet nisl. Vivamus hendrerit orci lectus. Maecenas nec nunc nec nisl scelerisque congue in nec justo. Maecenas nec dui in lorem tempus

What just happened?

Let's pick that last bit of code apart to understand what's happening there.

We're already familiar with this:

```
$(document).ready();
```

That's the ready method of the jQuery object being called on the document. That means that we'll run our code as soon as the document is ready. As usual, we've told jQuery what should happen as soon as the document is ready by passing a function to this method:

```
$(document).ready(function(){
   //our code will go here
});
```

So the only really new thing we have to look at is the line of code we wrote inside the function:

```
$('#scrolling').jScrollPane();
```

But even this we can understand at least a bit. We know that `$('#scrolling')` will select the item on the page with the `id` of scrolling. Remember, we wrapped `<div id="scrolling"></div>` around the paragraphs of text that we wanted to scroll. Then we used a couple lines of CSS to limit the height of the `#scrolling div` and show the browser's scrollbar.

Then we can see that we're calling the `jScrollPane()` method. Most jQuery plugins will work this way – by adding a new method that you can call. How do you know what the new method is named? You'll usually find it in the documentation for the plugin. jScrollPane is exceptionally well documented with piles of examples for you to pick apart, learn from, and modify.

Adding arrow controls

Okay, now that we've got the basics of using plugins under our belts, now we can take a look at how we can take it further.

Time for action – adding up and down arrows

Let's add top and bottom buttons to our scrollbars so our scrollbars look and behave more like native scrollbars.

1. Let's go back to that line of code in our `scripts.js` file where we called the `jScrollPane()` method to create the custom scrollbars:

   ```
   $('#scrolling').jScrollPane();
   ```

Remember how we could pass things to methods and functions by putting them inside the parentheses? We had the following example:

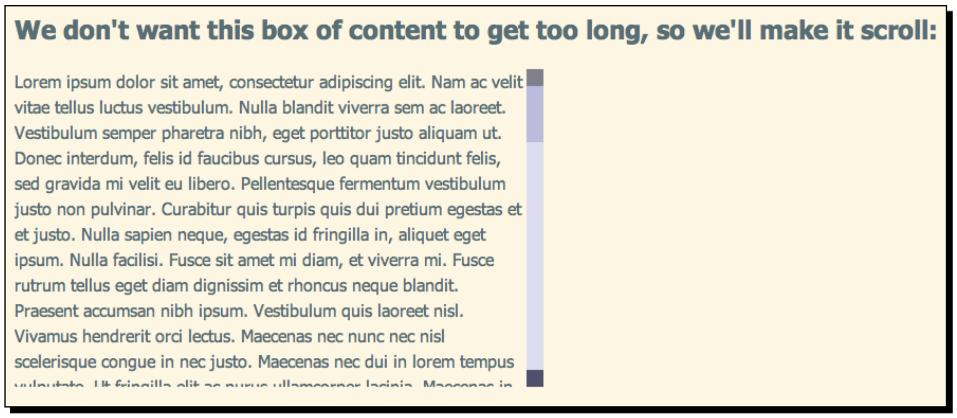
```
dog.eat('bacon');
```

where we wanted to say that the dog was eating bacon. So, in JavaScript-speak we passed bacon to the eat method of the dog.

Well, in this case, we can pass a set of options to the `jScrollPane` method to control how our scrollbars look and act. We want to show the top and bottom arrows on our scrollbars, and we can do that by setting the `showArrows` option to `true`. We just have to make a simple modification to our line of code as follows:

```
$('#scrolling').jScrollPane({showArrows:true});
```

2. Now when you refresh the page, you'll see boxes at the top and bottom of your scrollbars, just where top and bottom arrows would appear.



We don't want this box of content to get too long, so we'll make it scroll:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam ac velit vitae tellus luctus vestibulum. Nulla blandit viverra sem ac laoreet. Vestibulum semper pharetra nibh, eget porttitor justo aliquam ut. Donec interdum, felis id faucibus cursus, leo quam tincidunt felis, sed gravida mi velit eu libero. Pellentesque fermentum vestibulum justo non pulvinar. Curabitur quis turpis quis dui pretium egestas et et justo. Nulla sapien neque, egestas id fringilla in, aliquet eget ipsum. Nulla facilisi. Fusce sit amet mi diam, et viverra mi. Fusce rutrum tellus eget diam dignissim et rhoncus neque blandit. Praesent accumsan nibh ipsum. Vestibulum quis laoreet nisl. Vivamus hendrerit orci lectus. Maecenas nec nunc nec nisl scelerisque congue in nec justo. Maecenas nec dui in lorem tempus

If you click on these boxes, you'll see that they behave just like the up and down arrows on a regular scrollbar. They're just a little plain – we can style those up with some CSS to look any way we'd like.

What just happened?

We set the `showArrows` option of the `jScrollPane` method to `true`. There's a rather long list of advanced options available with this plugin, but luckily, we don't have to learn or know them all to be able to make good use of it.

How do we know that there's a `showArrows` option? We'll find it in the documentation for the plugin. Once you get better at understanding JavaScript and jQuery, you'll be able to read the plugin files themselves to see what options and methods the plugin is providing for you.

To pass one option to a method, you'll wrap it in curly braces. Then you'll type the name of the option you're setting (in this case, `showArrows`), then a colon, and then the value that you're setting the option to (in this case, `true` to show the arrows). Just like we did before:

```
$('#scrolling').jScrollPane({showArrows:true});
```

If you want to pass more than one option to a method, you'll do everything the same, except you'll need to put a comma between the options. For example, if I wanted to add a little breathing room between my text and the scrollbar, I could do that by setting a value for the `verticalGutter` option:

```
$('#scrolling').jScrollPane({ showArrows:true,verticalGutter:20});
```

Now, you can see that if I were setting a dozen or more options, this line of code would get long and hard to read. For that reason, it's common practice to break options out onto separate lines as follows:

```
$('#scrolling').jScrollPane({
    showArrows:         true,
    verticalGutter:     20
});
```

You can see that the content and order are the same, only this example is easier for a human being to read and understand. A computer doesn't care one way or the other.

Be careful not to add an extra comma after the last option/value pair. Most browsers will handle that error gracefully, but Internet Explorer will throw an error and your JavaScript won't work.

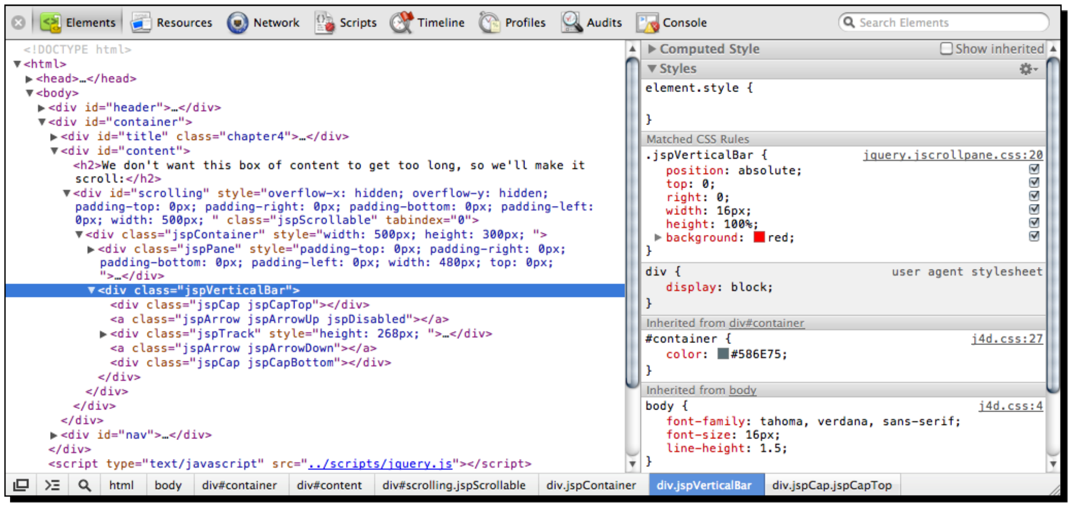
Customizing the Scrollbar Style

Now we've got top and bottom buttons on our scrollbars, so let's make them look just the way we want. We can do this by writing our own CSS to style the scrollbars.

If you've spent any time at all debugging CSS, then you already know about the tools available to you in your favorite browser to do so. Just in case you haven't, I highly recommend you take a look at the Firebug extension for Firefox, or the developer tools built into Opera, Chrome, Safari, and IE9. A quick Google search for 'your browser *developer tools tutorial*' should yield plenty of results where you can learn how to take advantage of what these tools have to offer.

If you're using an older version of IE, then take a look at the **Debug Bar** program available as an extension to IE that will be helpful for troubleshooting problems with CSS. It is free for personal use.

I tend to use Google Chrome when I'm developing new pages. To access the developer tools in Chrome, click the wrench icon at the far right of the toolbar, then select **Tools** | **Developer Tools**. Here's an example of the CSS information I can get by using the built-in tools:



```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▶<div id="header">…</div>
    ▼<div id="container">
      ▶<div id="title" class="chapter4">…</div>
      ▼<div id="content">
          <h2>We don't want this box of content to get too long, so we'll make it
          scroll:</h2>
        ▼<div id="scrolling" style="overflow-x: hidden; overflow-y: hidden;
          padding-top: 0px; padding-right: 0px; padding-bottom: 0px; padding-left:
          0px; width: 500px; " class="jspScrollable" tabindex="0">
          ▼<div class="jspContainer" style="width: 500px; height: 300px; ">
            ▶<div class="jspPane" style="padding-top: 0px; padding-right: 0px;
              padding-bottom: 0px; padding-left: 0px; width: 480px; top: 0px;
              ">…</div>
            ▼<div class="jspVerticalBar">
                <div class="jspCap jspCapTop"></div>
                <a class="jspArrow jspArrowUp jspDisabled"></a>
              ▶<div class="jspTrack" style="height: 268px; ">…</div>
                <a class="jspArrow jspArrowDown"></a>
                <div class="jspCap jspCapBottom"></div>
              </div>
            </div>
          </div>
        </div>
      ▶<div id="nav">…</div>
      </div>
      <script type="text/javascript" src="../scripts/jquery.js"></script>
```

```
element.style {

}

.jspVerticalBar {                jquery.jscrollpane.css:20
   position: absolute;                                 ✓
   top: 0;                                             ✓
   right: 0;                                           ✓
   width: 16px;                                        ✓
   height: 100%;                                       ✓
 ▶ background: 🟥 red;                                 ✓
}

div {                              user agent stylesheet
   display: block;
}

#container {                                   j4d.css:27
   color: ■#586E75;
}

body {                                          j4d.css:4
   font-family: tahoma, verdana, sans-serif;
   font-size: 16px;
   line-height: 1.5;
}
```

On the left, you can see the DOM for my document – all the HTML elements that make up the document tree. I can interact with it – each node can be expanded or collapsed to show or hide the elements that are nested inside. In this case, the highlighted element is the container for our jScrollPane scrollbar.

On the right, I can see the CSS that applies to the element I've selected on the left. I can also see which file that particular CSS appears in, and on what line. In this case, most of the CSS that's styling my scrollbar container can be found on line 20 of the `jquery.jscrollpane.css` file.

Digging into the DOM and CSS this way is a quick and easy way to figure out which lines of CSS we need to modify to get the appearance that we want.

We have a couple of options for customizing the CSS for the scrollbars. Either we can modify the `jquery.jscrollpane.css` file directly, or we can copy those styles to our own stylesheet and make the changes there. It's a matter of personal preference, but if you opt to modify the `jquery.jscrollpane.css` file directly, as I'm going to do here, then I highly recommend you to make a separate copy of it to keep so that you can refer to it or easily restore it without having to re-download it again.

Time for action – adding our own styles

1. Open `jquery.jscrollpane.css`. Around line 56, you'll find the CSS that styles `.jspTrack`. This is the track for our scrollbar – the background area you might say. The default style for it is a pale lavender color.

```
.jspTrack
{
	background: #dde;
	position: relative;
}
```

We don't want to mess with the position, since our scrollbar is relying on that to work correctly, but you can feel free to change the background color to any color, gradient, or image you'd like. I'm going to make mine pale pink:

```
.jspTrack
{
	background: #fbebf3;
	position: relative;
}
```

```
The next style I'd like to change is for .jspDrag. This is the
actual scrollbar handle. I'm going to make it bright pink:
.jspDrag
{
	background: #D33682;
	position: relative;
	top: 0;
	left: 0;
	cursor: pointer;
}
```

2. Next, I'll tackle the top and bottom buttons. I have not only a default style, but also a disabled style. For example, when the scroll area is all the way to the top the top button is disabled since I can't possibly scroll any higher. If I examine the buttons with my developer tools, I can also see that there's an additional class name on the buttons that's not styled in the default CSS – the top button has a class of `jspArrowUp` and the bottom button has a class of `jspArrowDown`. That will let me set a different style for the up and down buttons – I'm going to use an image of an upward pointing arrow as a background for the top arrow, and a downward-pointing arrow for the bottom button to make their function clear to my site visitors.

Here's my CSS for styling those:

```
.jspArrow
{
       text-indent: -20000px;
       display: block;
       cursor: pointer;
}

.jspArrow.jspDisabled
{
       cursor: default;
       background-color: #999;
}

.jspArrowUp
{
  background: #d6448b url(../images/arrows.png) 0 0 no-repeat;
}

.jspArrowDown
{
  background: #d6448b url(../images/arrows.png) 0 -16px no-repeat;
}
```

What just happened?

Now when you refresh the browser, you can see that the scrollbars are styled pink – just the way I wanted them. We modified the CSS that was supplied by the plugin developer to make the scrollbars appear just the way we wanted. We were able to use the developer tools built into our browser to target the file and line numbers of the code that needed to be updated to change the appearance of the scrollbars.

Have a go hero – style the scrollbars the way you want

Now, you might not care for bright pink scrollbars, and you might think my example is a little bit plain, and you'd be right. But you can get creative with background colors, gradients, images, rounded corners, and more, to style your scrollbars just the way you'd like. You can mimic the scrollbars of your favorite operating system so that all of your site visitors see them the way you like, or you can create an entirely new style. Experiment with the CSS to create your own scrollbar style.

Smooth scrolling

jScrollPane is a mature and full-featured plugin. If you poke through the examples and documentation, you'll find all kinds of fun options to play with. I'll walk you through setting up one of my favorites: animated scrolling inside the scrollable area.

Time for action – setting up smooth scrolling

You could place any kind of content you'd like inside a scrollable area—a list of news stories, a gallery of photos, or a long article with several sections, headings, and subheadings, for example. Here's how you can set up a control to smoothly scroll from one section to another:

1. The first thing we'll need to do is assign an ID to each of our paragraphs. I have five paragraphs of lorem ipsum in my scrollable area, so I'm going to assign them `id`s of `para1`, `para2`, `para3`, `para4`, and `para5`. You can choose whatever `id`s you like, but keep in mind that an `id` cannot begin with a number. So now my code looks like this (I've truncated the text to save space):

   ```
   <div id="scrolling">
          <p id="para1">Lorem ipsum...</p>
          <p id="para2">...</p>
          <p id="para3">...</p>
          <p id="para4">...</p>
          <p id="para5">...</p>
   </div>
   ```

2. Now, let's add in some internal links above our scrollable area to jump to each of these paragraphs. After the heading and before the scrollable area, add the following code:

   ```
   <h2>We don't want this box of content to get too long, so we'll
   make it scroll:</h2>
   <p>Scroll to a paragraph:
          <a href="#para1">1</a>,
          <a href="#para2">2</a>,
          <a href="#para3">3</a>,
          <a href="#para4">4</a>,
          <a href="#para5">5</a>
   </p>
   <div id="scrolling">
   ```

3. If we have JavaScript disabled, these links work; they will scroll the scrollable area down to the paragraph in question making it visible to our site visitor. But we want them to work with our fancy custom scrollbars. So we just have to pass a new option to our jScrollPane method:

```
$(document).ready(function(){
       $('#scrolling').jScrollPane({
              showArrows:                        true,
              verticalGutter:                    30,
              hijackInternalLinks:       true
       });
});
```

   This new option is to keep the browser from attempting its default behavior when the internal links are clicked. Refresh the page, and try out the links to the paragraphs.

4. It works, but it's not exactly pretty, and it can be a little disconcerting when that scrollable area jumps suddenly like that—our site visitor might not realize exactly what's happened. Let's make it obvious by smoothly animating that jump to the different paragraphs. All we have to do is add another option to our code:

```
$(document).ready(function(){
       $('#scrolling').jScrollPane({
              showArrows:         true,
              verticalGutter:            30,
              hijackInternalLinks:       true,
              animateScroll:             true
       });
});
```

   Now, when you refresh the page and click the paragraph links, you'll see that the scrollable area smoothly scrolls to the proper paragraph. It's easy to understand what's happening and where you are on the page and in the scrollable area.

What just happened?

We took advantage of one of the features of the jScrollPane plugin and made smooth scrolling to any bit of content inside our scrollable container possible. The options and values available to us are all documented in the plugin's documentation and examples. You can see how easy it was to customize this plugin to add this nice bit of behavior, thanks to the plugin author's hard work in making tough stuff easy for us.

Summary

Phew! This was quite a chapter. We learned about jQuery plugins, how to use them, and how to use the options they make available to customize them. We learned about dependencies and inserting multiple scripts into our file in the correct order. We used Kelvin Luck's excellent jScrollPane plugin to replace our boring operating system scrollbars with fancy custom ones of our own design. And the bonus is, they work just like browser scrollbars – our site visitors can click on the track, on the up and down buttons, they can drag the handle, or they can use their mousewheel to navigate up and down the scrollable areas we've set up. It's a win for both aesthetics and usability.

Finally, we learned how to smoothly scroll to an anchor inside the scrollable area – this allows our site visitors to easily get to individual bits of content inside the scrollable area, and communicates what's happening clearly.

Next up, we'll take a look at overriding the browser's default tooltips with nicely designed tooltips of our own and we'll learn how to make them work even harder for us by adding extra content.

5
Creating Custom Tooltips

Now that we've seen how powerful plugins are and how easy they make advanced functionality, let's see how we can take advantage of another plugin to make custom tooltips.

Browsers automatically create tooltips when you include the title attribute—usually on a link or an image. When your site visitor hovers their mouse cursor over the item or moves focus to the item by tabbing to it, the tooltip will appear—usually as a small yellow box that appears to be floating over the page. Tooltips are a great way to add a little additional information to your page. Screen reader software reads out tooltip text for site visitors with disabilities who are using assistive technology, making them useful for enhancing accessibility. Furthermore, title attributes on images and links can help search engines index your content more effectively.

In this chapter, we'll learn:

- How to use Craig Thompson's qTip plugin to replace the browser's default tooltips
- How to customize the appearance of the qTip tooltips
- How to enhance a navigation bar with custom tooltips
- How to display Ajax content in custom tooltips

Simple custom text tooltips

I hope I've convinced you that `title` attributes are great for enhancing both the usability and accessibility of your site. The only problem with tooltips is that they can't be customized in any way. Each browser has its own style of tooltip and that style is not customizable via CSS. Sometimes this is fine, but sometimes it's nice to have more control over the appearance of tooltips.

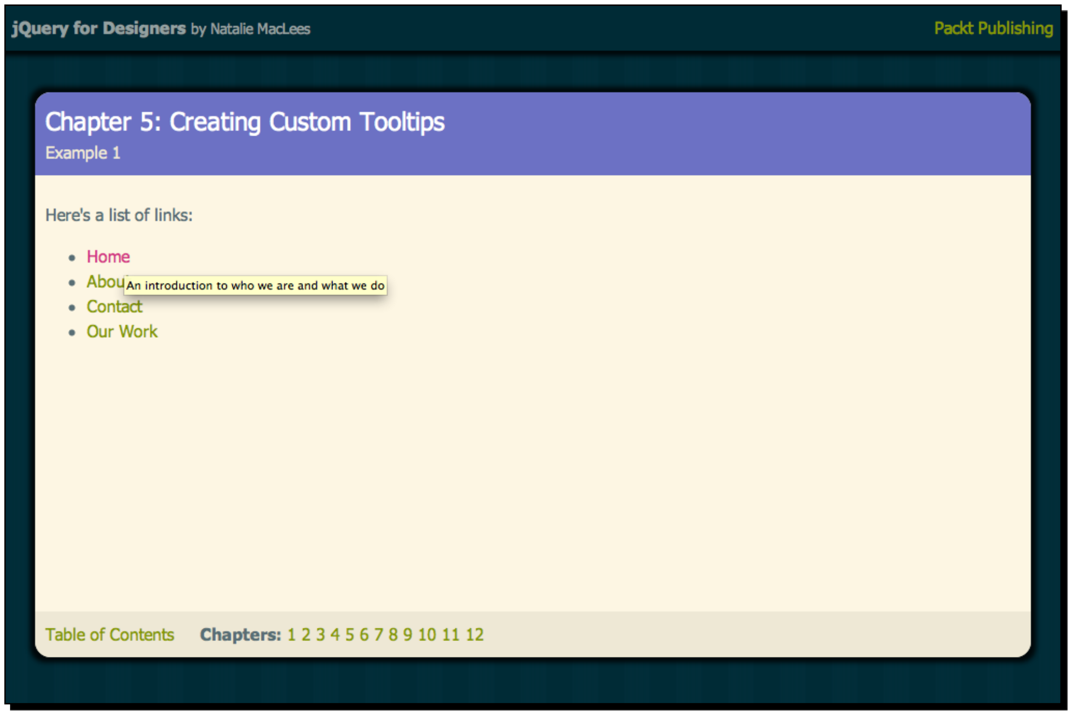
Time for action – simple text tooltips

We'll start off working with tooltips by making a simple replacement for the browser's default tooltip that we can style any way we'd like:

1. Set up a basic HTML file and associated files and folders like we did in *Chapter 1, Designer, Meet jQuery*. Our HTML file should contain a set of links that each have a `title` attribute like this:

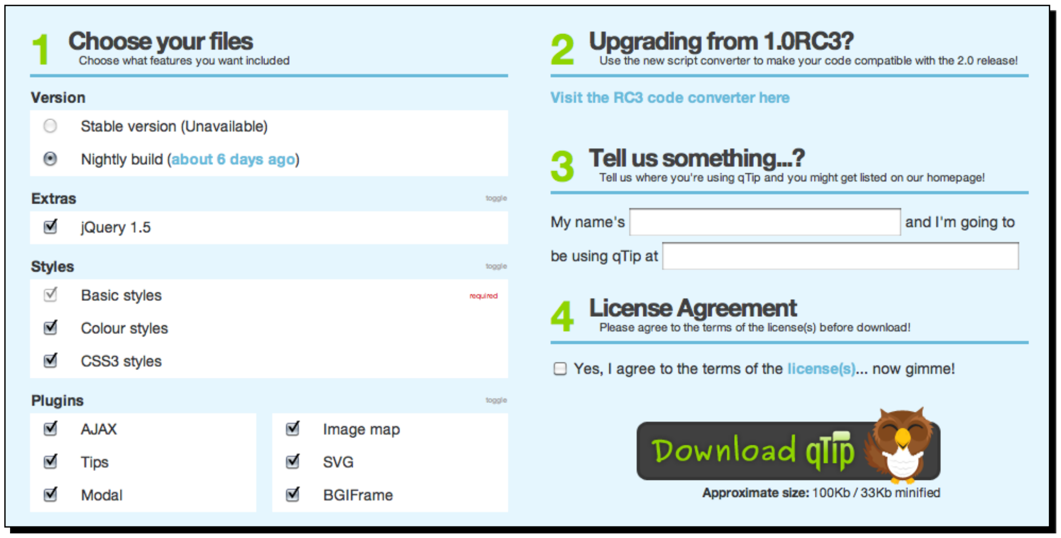
```
<p>Here's a list of links:</p>
<ul>
   <li><a href="home.html" title="An introduction to who we are
and what we do">Home</a></li>
   <li><a href="about.html" title="Learn more about our
company">About</a></li>
   <li><a href="contact.html" title="Send us a message. We'd love
to hear from you!">Contact</a></li>
   <li><a href="work.html" title="View a portfolio of the work
we've done for our clients">Our Work</a></li>
</ul>
```

2. Open that page in a browser and move your mouse over the links. You'll see the text contained in the `title` attribute shown in a tooltip. Exactly where the tooltip appears and exactly what it looks like will depend on your browser, but here's how it looks in mine (Google Chrome on Mac OS):


jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 5: Creating Custom Tooltips
Example 1
Here's a list of links:
Home
Abou
An introduction to who we are and what we do
Contact
Our Work
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

3. Now, let's spruce that up a bit by replacing the default browser tooltip with our own styled one. First, we'll need to download Craig Thompson's qTip plugin. It's available from `http://craigsworks.com/projects/qtip2`. His site has a list of features, several sample demos, the documentation you'll need to learn to use the plugin, a forum where you can get help, and the files needed are available for download. Head to the download page, and you'll see a checklist of options to help you download the right version.

Let's walk through this page one section at a time:



1 Choose your files
Choose what features you want included

Version
- ○ Stable version (Unavailable)
- ● Nightly build (about 6 days ago)

Extras — toggle
- ✓ jQuery 1.5

Styles — toggle
- ✓ Basic styles — required
- ✓ Colour styles
- ✓ CSS3 styles

Plugins — toggle
- ✓ AJAX
- ✓ Tips
- ✓ Modal
- ✓ Image map
- ✓ SVG
- ✓ BGIFrame

2 Upgrading from 1.0RC3?
Use the new script converter to make your code compatible with the 2.0 release!

Visit the RC3 code converter here

3 Tell us something...?
Tell us where you're using qTip and you might get listed on our homepage!

My name's ______ and I'm going to be using qTip at ______

4 License Agreement
Please agree to the terms of the license(s) before download!

☐ Yes, I agree to the terms of the license(s)... now gimme!

Download qTip

Approximate size: 100Kb / 33Kb minified

4. **Step 1** gives us a number of options for downloading the script. In the section titled **Version**, I'm going to select **Stable** version so that I get the latest version of the script that has been tested thoroughly. Those wanting to experiment with and test the plugin as the developer works on it, can select the nightly build.

5. In the **Extras** section, I'm going to uncheck **jQuery 1.5** since I have already downloaded jQuery and attached it to my project. If you're starting a new project and haven't yet downloaded jQuery, you can leave this checked to download jQuery simultaneously with the plugin.

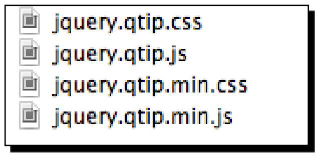
6. In the **Styles** section, I'm going to leave all three sets of styles selected, since I want as many options as possible for styling my tooltips. Likewise, I'm going to leave all options selected in the **Plugins** section since I'll be working on a variety of different types of tooltips and taking advantage of these different features. If you wanted to simply create simple text-based tooltips, you could uncheck all of these extras and get a much smaller download file. These extras are only needed if you're going to be taking advantage of the extra features. It's a nice feature of this plugin that we can pick and choose just the functionality we want in order to keep our JavaScript files as small as possible.

7. **Step 2** offers an automatic converter for anyone who is updating their code that might have previously used an earlier version of the plugin. We can ignore this step since we're newcomers to the qTip plugin.

8. **Step 3** gives us the opportunity to tell the plugin developer about our site that uses the plugin in exchange for a chance to be featured on the plugin's home page gallery. Since we're only doing some practice exercises in this chapter, we won't use this now, but it may be something for you to consider for your own projects later on.

9. **Step 4** requires us to accept the terms of the license. This plugin is licensed under the open source MIT and GPLv2 licenses, which makes it free for us to use, modify and even redistribute the code, provided the license or link to the license is included in the files. The license is already included in the plugin files when you download them, so as long as you don't edit those files to remove the license, you'll be fine.

10. Finally, we can click the **Download qTip** button, and your browser will download a ZIP file for you. Unzip it and examine its contents. Inside, we'll find two CSS files and two JavaScript files. (You might have an extra JavaScript file if you elected to download jQuery as well as the plugin script).



jquery.qtip.css
jquery.qtip.js
jquery.qtip.min.css
jquery.qtip.min.js

11. Let's start with the two CSS files. We have `jquery.qtip.css` and `jquery.qtip.min.css`. These two files have exactly the same content. The difference between them is that the second file is minified, making it smaller and ideal for use in production. The other file is the development version that we could easily edit ourselves or use as an example if we wanted to write our own styles for our tooltips instead of using the prebuilt styles. You'll select one of the files and attach it to your page. In this example, I'm going to use the minified version of the file to keep the file as small as possible since I don't want to write my own styles at this point. Copy `jquery.qtip.min.css` to your own `styles` folder, and then attach the file to your HTML document in the `<head>` section:

```
<head>
   <title>Chapter 5: Creating Custom Tooltips</title>

   <link rel="stylesheet" href="styles/jquery.qtip.min.css"/>
   <link rel="stylesheet" href="styles/styles.css"/>
</head>
```

I'm attaching the qTip stylesheet before my own `styles.css` to make it easier for me to override styles in the qTip stylesheet if I want to.

12. Next, let's look at the JavaScript files. We have `jquery.qtip.js` and `jquery.qtip.min.js`. Just like the CSS files, these are two different versions of the same file, and we simply have to pick one and attach it to our HTML document. The first file, `jquery.qtip.js`, is the development version of the file, and the largest file at 94K. The second file is minified and weighs in at only 41K. Since we don't need to edit the plugin and are going to be using it as is, let's select the minified version. Copy `jquery.qtip.min.js` to your own `scripts` folder and attach it at the bottom of your HTML file, in between jQuery and our own `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.qtip.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

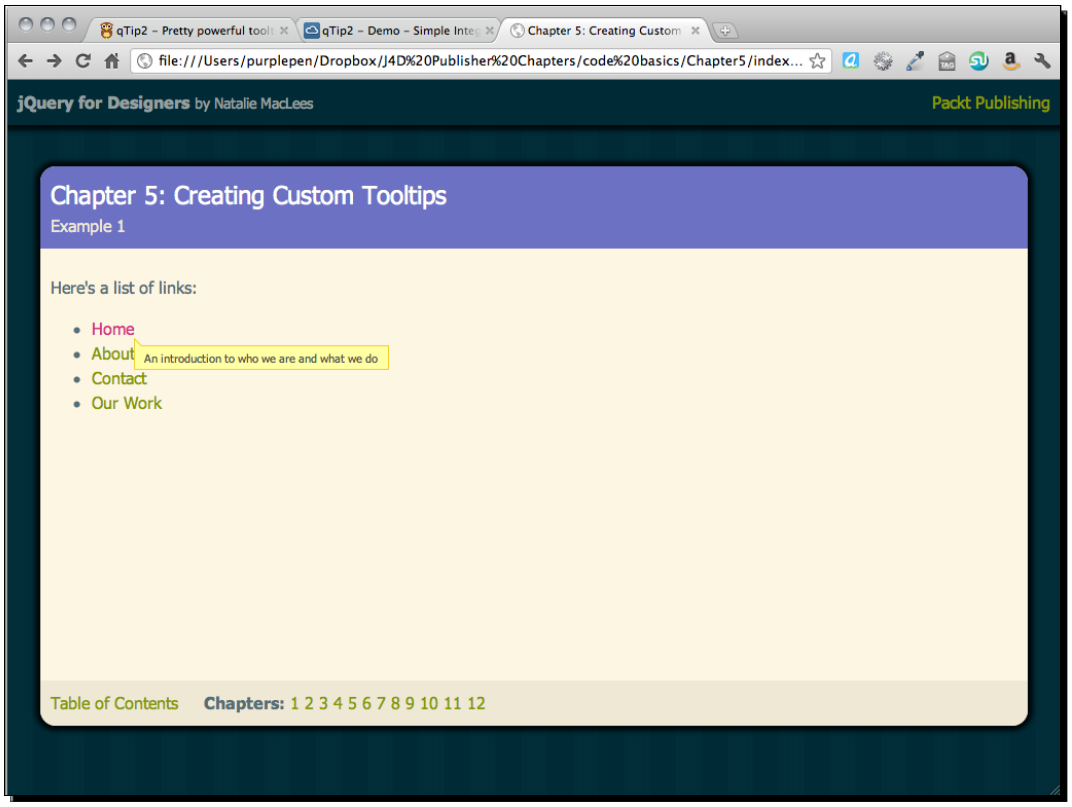
13. The last thing we need to do is call the plugin code. Open your `scripts.js` file and add the document ready statement and function:

```
$(document).ready(function(){
});
```

14. Inside the function, select all links in the document that have `title` attributes and call the `qtip()` method on those links:

```
$(document).ready(function(){
      $('a[title]').qtip();
   });
```

15. Now, when you view the page in the browser and move your mouse over the links with `title` attributes, you'll see the qTip-styled tooltips instead of the browser's default tooltips:


qTip2 – Pretty powerful tool
qTip2 – Demo – Simple Integ
Chapter 5: Creating Custom
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/code%20basics/Chapter5/index...
jQuery for Designers by Natalie MacLees
Packt Publishing

Chapter 5: Creating Custom Tooltips

Example 1

Here's a list of links:

- Home
- About An introduction to who we are and what we do
- Contact
- Our Work

Table of Contents **Chapters:** 1 2 3 4 5 6 7 8 9 10 11 12

Even better, these tooltips will appear with this same style, no matter which browser and operating system we're using.

What just happened?

We downloaded the qTip plugin and attached one CSS file and one JavaScript file to our HTML document. Then we added just a couple of lines of jQuery code to activate the custom tooltips.

We selected all the link elements on the page that had a `title` attribute. We did this by taking advantage of jQuery's attribute selectors:

```
$('a[title]')
```

Putting `title` in brackets after our element selector means that we only want those links on the page that have a `title` attribute.

Once we've selected those links, all that's left to do is to call the `qtip()` method that the qTip plugin provided for us. The `qtip()` method takes care of all the actions that need to be done to replace the default tooltip with a custom one. But what if we want to use some of the other styles included with qTip?

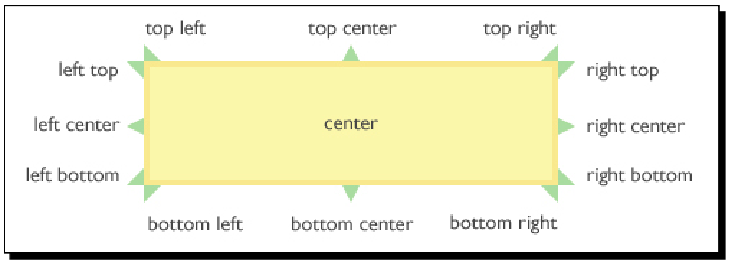
Customizing qTip's appearance

You've undoubtedly noticed that the top-left corner of the qTip aligns with the bottom-right corner of the link when the mouse hovers over the link, and that the tooltip appears as a yellow box with a small arrow on the side. The qTip plugin offers lots of options for customizing where the tooltip appears and what it looks like and it does so in a straightforward and easy to understand way.

Time for action – customizing qTips

Let's take a look at the options we have for customizing the appearance of qTip's tooltips:

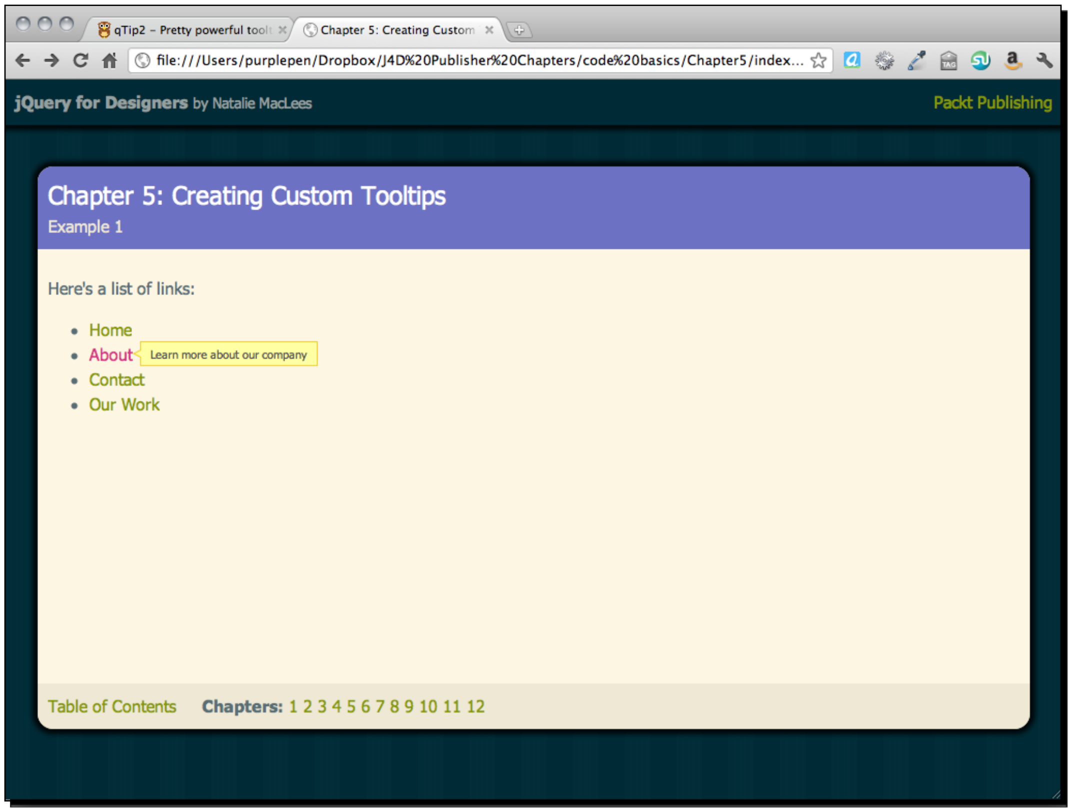
1. Let's say that we want to change the position of the tooltip. qTip gives us plenty of options for positioning our tooltips on our page.
2. We can match up any of these points on the tooltip to any of these points on the link:



3. In this example, we'll match up the middle of the link's right side with the middle of the tooltip's left side, so that the tooltip appears directly to the right of the link. We simply need to pass some additional information to the `qTip()` method. We'll keep working with the files we set up in the last example. Open your `scripts.js` file and pass this additional information to the `qtip()` method:

```
$('a[title]').qtip({
   position: {
      my: 'center left',
      at: 'center right'
}
});
```

The developer's goal was for this to make sense in plain language. Speaking from the tooltip's point of view, we're going to align my center-left at the link's center-right. As you can see when we refresh the page in the browser, the tooltip now appears directly to the right of the link.


qTip2 – Pretty powerful tool
Chapter 5: Creating Custom
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/code%20basics/Chapter5/index...
jQuery for Designers by Natalie MacLees
Packt Publishing

Chapter 5: Creating Custom Tooltips
Example 1

Here's a list of links:

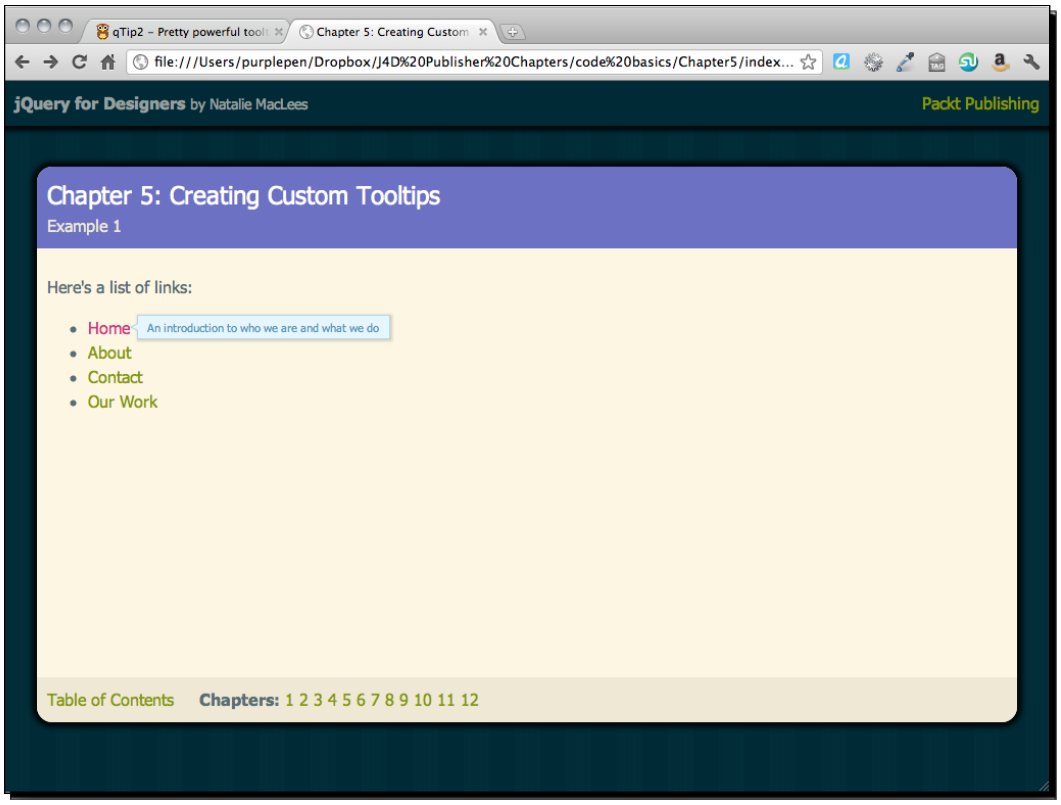
- Home
- About Learn more about our company
- Contact
- Our Work

Table of Contents Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

4. In addition to changing the position of the tooltip, we can change the appearance of the tooltip itself. The CSS included with the plugin includes several color schemes and styles. These different colors and styles are applied by adding CSS classes to our tooltip. Let's take a look at how we add these CSS classes.

```
$('a[title]').qtip({
   position: {
      my: 'center left',
      at: 'center right'
   },
   style: {
      classes: 'ui-tooltip-blue'
   }
});
```

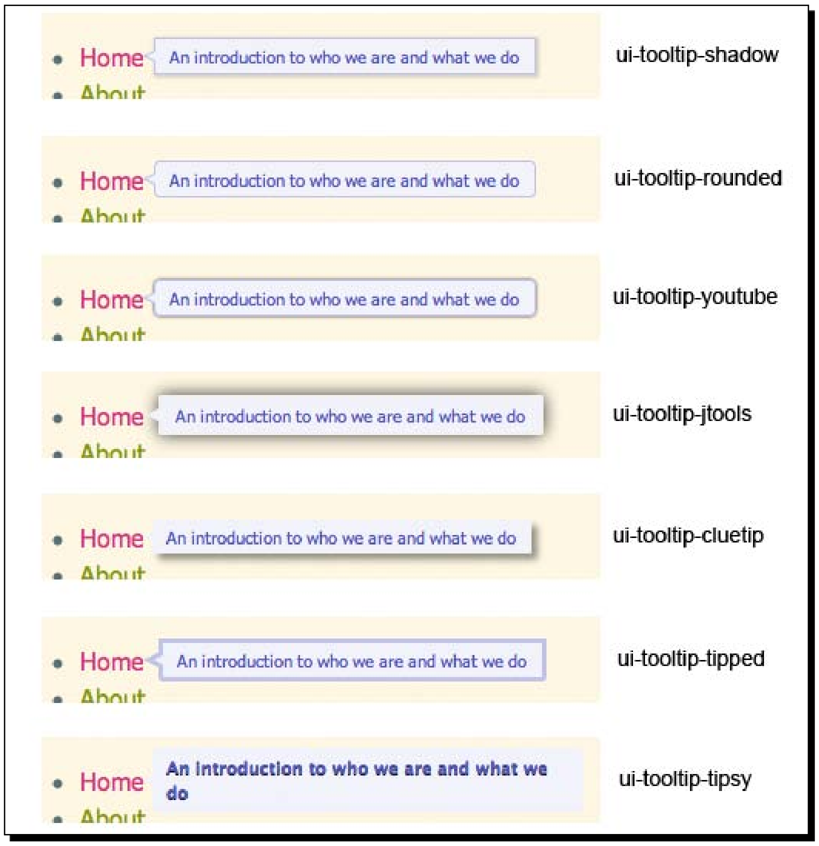
   Now when we view our tooltip in the browser, we see that it's blue:


qTip2 – Pretty powerful tool
Chapter 5: Creating Custom
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/code%20basics/Chapter5/index...
jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 5: Creating Custom Tooltips
Example 1
Here's a list of links:
Home
An introduction to who we are and what we do
About
Contact
Our Work
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

5. The color schemes provided with qTip include:
   - `ui-tooltip` (the default yellow color scheme)
   - `ui-tooltip-light` (black text on a white tooltip)
   - `ui-tooltip-dark` (white text on a dark grey tooltip)
   - `ui-tooltip-red`
   - `ui-tooltip-green`
   - `ui-tooltip-blue`

   You can add any one of these classes to your tooltips to adjust the color scheme.

6. For CSS3-capable browsers, qTip also offers some fancier styles. These styles won't be seen in browsers that don't support the CSS3 specification, but in most cases, that should be fine. These styles can be considered as progressive enhancement for the browsers that can display them. Site visitors using a less capable browser will still be able to see and read the tooltips without any trouble. They just won't see the fancier styles applied. The available styles are as follows:



7. Just like with the color schemes, we can take advantage of these styles by adding CSS classes to our tooltips. Multiple CSS classes can be added to a tooltip like so:

```
$('a[title]').qtip({
   position: {
      my: 'center left',
      at: 'center right'
   },
   style: {
      classes: 'ui-tooltip-blue ui-tooltip-shadow'
   }
});
```

This code creates a tooltip that is blue and has a shadow.

What just happened?

We saw how we can pass position and style values to the qTip method to customize the appearance of our tooltips. We learned the color schemes and styles that are included with the qTip plugin, and learned how we can use those styles in our own pages to customize the qTip tooltips.

Custom styles for tooltips

We can also write our own color schemes and styles for our tooltips if none of the available options are quite right for our site.

Time for action – writing custom tooltip styles

Let's take a look at how we can write our own custom styles for qTip's tooltips by writing a new purple color scheme:

1. We'll get started by examining the CSS that codes up the red tooltip style that comes with qTip. You'll find this bit of CSS inside the `jquery.qtip.css` file that was included with the qTip download. Here are all the CSS styles that affect the red tooltips:

```
/*! Red tooltip style */
.ui-tooltip-red .ui-tooltip-titlebar,
.ui-tooltip-red .ui-tooltip-content{
   border-color: #D95252;
   color: #912323;
}
```

```
    .ui-tooltip-red .ui-tooltip-content{
        background-color: #F78B83;
    }
    .ui-tooltip-red .ui-tooltip-titlebar{
        background-color: #F06D65;
    }
    .ui-tooltip-red .ui-state-default .ui-tooltip-icon{
        background-position: -102px 0;
    }
    .ui-tooltip-red .ui-tooltip-icon{
        border-color: #D95252;
    }
    .ui-tooltip-red .ui-tooltip-titlebar .ui-state-hover{
        border-color: #D95252;
}
```

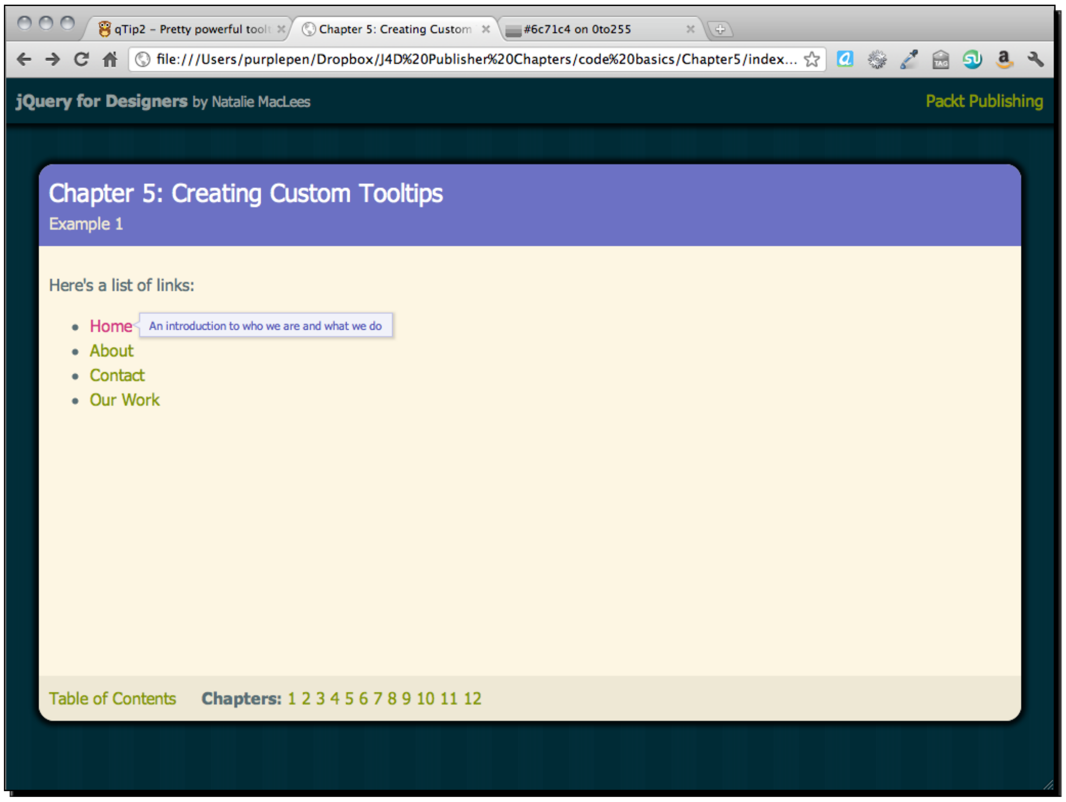
2. From examining this CSS, we can see that all we need to do to create a new color scheme is to create a new class name and four shades of purple to create a new style. Here's the CSS for my purple color scheme. Open your `styles.css` file and add these styles:

```
/*! Purple tooltip style */
.ui-tooltip-purple .ui-tooltip-titlebar,
.ui-tooltip-purple .ui-tooltip-content{
    border-color: #c1c3e6;
    color: #545aba;
}
    .ui-tooltip-purple .ui-tooltip-content{
        background-color: #f1f2fa;
    }
    .ui-tooltip-purple .ui-tooltip-titlebar{
        background-color: #d9daf0;
    }
    .ui-tooltip-purple .ui-state-default .ui-tooltip-icon{
        background-position: -102px 0;
    }
    .ui-tooltip-purple .ui-tooltip-icon{
        border-color: #c1c3e6;
    }
    .ui-tooltip-purple .ui-tooltip-titlebar .ui-state-hover{
        border-color: #c1c3e6;
}
```

3. Now, to take advantage of our new purple tooltip style, we simply have to adjust our jQuery code to add the newly created `ui-tooltip-purple` class to our tooltips. Open `scripts.js` and adjust the classes being added to the tooltips:

```
$('a[title]').qtip({
   position: {
      my: 'center left',
      at: 'center right'
   },
   style: {
      classes: 'ui-tooltip-purple'
   }
});
```

   Now, when you preview the link in the browser, you will see a purple tooltip, as shown in the following screenshot:


qTip2 – Pretty powerful tool
Chapter 5: Creating Custom
#6c71c4 on 0to255
file:///Users/purplepen/Dropbox/J4D%20Publisher%20Chapters/code%20basics/Chapter5/index...
jQuery for Designers by Natalie MacLees
Packt Publishing

Chapter 5: Creating Custom Tooltips
Example 1

Here's a list of links:

- Home An introduction to who we are and what we do
- About
- Contact
- Our Work

Table of Contents Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

What just happened?

Using one of the CSS classes provided with qTip, we wrote our own custom style and applied it to our tooltips. You can use any CSS styles you'd like to create a custom appearance for the qTip tooltips. There's virtually no limit to the possibilities for styles when you start mixing in color and font choices, background images, border styles, and so on.

Have a go hero – create a tooltip of your own design

Try writing your own CSS class to style the tooltips. Try a new color scheme, new font styles and sizes, text shadows, box shadows—anything you can think of to make the tooltips match the design of a site or really stand out.

Enhancing navigation with tooltips

Once you know how to make custom tooltips, you'll find there are lots of possible uses for them. Let's take a look at enhancing a standard navigation bar with custom tooltips using the qTip plugin.

Time for action – building a fancy navigation bar

Let's take a look at how we can use custom-designed tooltips to add a little progressively enhanced punch to a basic navigation bar:

1. Let's start by setting up a basic HTML page with associated folders and files just as we did in *Chapter 1*, *Designer, Meet jQuery*. In the body of the document, include a simple navigation bar like this:

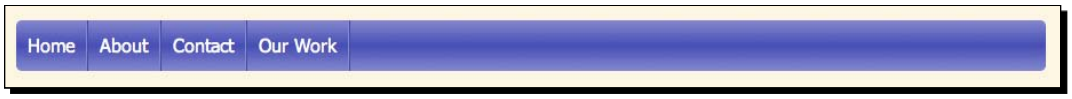
```
<ul id="navigation">      <li><a href="home.html" title="An
introduction to who we are and what we do">Home</a></li>
<li><a href="about.html" title="Learn more about our
company">About</a></li>
<li><a href="contact.html" title="Send us a message. We'd love to
hear from you!">Contact</a></li>
<li><a href="work.html" title="View a portfolio of the work we've
done for our clients">Our Work</a></li>
</ul>
```

2. Next, we'll add some CSS styles to our navigation bar. There's a lot of CSS here because I'm using a gradient as a background and it requires a lot of different CSS for different browsers right now. Add these lines of CSS to your `styles.css` file. If you prefer a different style, feel free to customize the CSS to suit your own taste:

```
#navigation {

   background: rgb(132,136,206); /* Old browsers */
   background: -moz-linear-gradient(top, rgba(132,136,206,1) 0%,
rgba(72,79,181,1) 50%, rgba(132,136,206,1) 100%); /* FF3.6+ */

   background: -webkit-gradient(linear, left top, left
bottom, color-stop(0%,rgba(132,136,206,1)), color-
stop(50%,rgba(72,79,181,1)), color-stop(100%,rg
ba(132,136,206,1))); /* Chrome,Safari4+ */

   background: -webkit-linear-gradient(top, rgba(132,136,206,1)
0%,rgba(72,79,181,1) 50%,rgba(132,136,206,1) 100%); /*
Chrome10+,Safari5.1+ */

   background: -o-linear-gradient(top, rgba(132,136,206,1)
0%,rgba(72,79,181,1) 50%,rgba(132,136,206,1) 100%); /* Opera11.10+
*/
   background: -ms-linear-gradient(top, rgba(132,136,206,1)
0%,rgba(72,79,181,1) 50%,rgba(132,136,206,1) 100%); /* IE10+ */

   filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#8488ce', endColorstr='#8488ce',GradientType=0 );
/* IE6-9 */

   background: linear-gradient(top, rgba(132,136,206,1)
0%,rgba(72,79,181,1) 50%,rgba(132,136,206,1) 100%); /* W3C */

   list-style-type: none;
   margin: 100px 20px 20px 20px;
   padding: 0;
   overflow: hidden;
   -webkit-border-radius: 5px;
   -moz-border-radius: 5px;
   border-radius: 5px;
}

#navigation li      {
   margin: 0;
   padding: 0;
```

```
        display: block;
        float: left;
        border-right: 1px solid #4449a8;
    }

    #navigation a{
        color: #fff;
        border-right: 1px solid #8488ce;
        display: block;
        padding: 10px;
    }

    #navigation a:hover {
        background: #859900;
        border-right-color: #a3bb00;
    }
```

3. Now we have a navigation bar horizontally across our page, like this:



4. I've included `title` attributes on my links and when I move my mouse over the navigation links, those are visible. I'd like to replace these boring browser default tooltips with friendly-looking conversation bubbles below my navigation.

5. Just like we did in the previous example, we're going to copy the qTip CSS and JavaScript to our own styles and scripts folders and attach them to the HTML document:

```
<!DOCTYPE html>
<html>
<head>
   <title>Chapter 5: Creating Custom Tooltips</title>
   <link rel="stylesheet" href="styles/jquery.qtip.min.css"/>

<script src="../scripts/jquery.js"></script>
<script src="scripts/jquery.qtip.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

6. Next, open your `scripts.js` file so we can call the `qtip()` method and pass in our customizations. We'll start off nearly the same way as last time, except that we'll use a different selector, since we only want to select links inside the navigation bar:

   ```
   $(document).ready(function(){
      $('#navigation a').qtip();
   });
   ```

   Now the default tooltips are replaced with qTip-styled tooltips.

7. Next, we're going to create our own style for the navigation tooltips, so we'll write some new CSS to make them look like speech bubbles. Add these styles to your `styles.css` file:

   ```
   .ui-tooltip-conversation .ui-tooltip-titlebar,
   .ui-tooltip-conversation .ui-tooltip-content{
      border: 3px solid #555;
      filter: none; -ms-filter: none;
   }

   .ui-tooltip-conversation .ui-tooltip-titlebar{
      background: #859900;
      color: white;
      font-weight: normal;
      font-family: serif;
      border-bottom-width: 0;
   }

   .ui-tooltip-conversation .ui-tooltip-content{
      background-color: #F9F9F9;
      color: #859900;
      -moz-border-radius: 9px;
      -webkit-border-radius: 9px;
      border-radius: 9px;
      padding: 10px;
   }

   .ui-tooltip-conversation .ui-tooltip-icon{
      border: 2px solid #555;
      background: #859900;
   }

   .ui-tooltip-conversation .ui-tooltip-icon .ui-icon{
         background-color: #859900;
         color: #555;
      }
   ```

8. Now that we've got a new CSS style for our tooltips ready to go, we just have to add this new class to the tooltips. Go back to `scripts.js` and add the new class to the JavaScript:

```
$('#navigation a').qtip({
   style: {
      classes: 'ui-tooltip-conversation'
   }
});
```

9. Next, let's position the speech bubbles so that they appear underneath each of the navigation links. In `scripts.js`, pass the position information to the `qtip()` method:

```
$('#navigation a').qtip({
   position: {
      my: 'top center',
      at: 'bottom center'
   },
   style: {
      classes: 'ui-tooltip-conversation',
      width: '150px'
   }
});
```

10. Now, we need to control the width of the tooltips so they don't appear too wide. We'll set the width to 150px:

```
$('#navigation a').qtip({
   position: {
      my: 'top center',
      at: 'bottom center'
   },
   style: {
      classes: 'ui-tooltip-conversation',
      width: '150px'
   }
});
```

11. Now the last thing we'll do is change the way the tooltips appear and disappear from the page. By default, the qTip plugin uses a very quick and subtle fade in and fade out. Let's change that, so that the tooltips slide into view and slide back out of view:

```
$('#navigation a').qtip({
   position: {
      my: 'top center',
      at: 'bottom center'
   },
   show: {
      effect: function(offset) {
         $(this).slideDown(300);
      }
   },
   hide: {
      effect: function(offset) {
         $(this).slideUp(100);
      }
   },
   style: {
      classes: 'ui-tooltip-conversation',
      width: '150px'
   }
});
```

12. Now when you view the page in a browser, you can see the conversation bubbles slide into view underneath each navigation link when you move your mouse over the link, and slide back out of view when you move your mouse off the link.

What just happened?

We reviewed how to create and attach a custom CSS style to qTip's tooltips and how to position the tooltip wherever you'd like it to appear. We also learned how to control the width of the tooltips to ensure we get a uniform size.

Then we saw how to override the default show and hide behaviors and replace them with custom animations. In this case, we used jQuery's `slideDown()` effect to show the tooltips. We passed a value of 300 to the `slideDown()` method, which means the animation will take 300 milliseconds to complete, or about a third of a second. I've found that if an animation takes much longer than that, site visitors get impatient waiting for it.

Next, we overrode the default hide behavior with jQuery's `slideUp()` method. I passed a value of 100, meaning the animation will complete rather quickly in about one-tenth of a second. When this animation runs, the site visitor has already decided to move on, so it's best to get the information out of their way as quickly as possible.

Showing other content in tooltips

So far we've seen how we can customize the appearance of qTip's tooltips, controlling their appearance, animation, and position. However, we've only used the tooltips to show text, namely the text we've placed inside a link's `title` attribute. We have a lot of more powerful options, though. We can load just about any content we'd like into our tooltips. We can also make sure the tooltips appear when an item is clicked instead of hovered over. Let's take a look at how we can load in content from another HTML page into our tooltips when we click on a link.

In this section, we'll dive into using Ajax for the first time. In case you aren't familiar, **Ajax** is a method for fetching some new content from the server and displaying it to the site visitor without having to completely refresh the page. Because the browser is only getting and displaying just the bit of information the site visitor needs, it's often much faster and snappier.

Just a quick note before we dive into Ajax for the first time. Modern browsers have several security rules for Ajax requests. You won't be able to simply view your ajaxified HTML files in a browser as we've been doing up until this point. In order to view the Ajax in action, you'll either have to upload your files to a server before viewing them, or you'll have to set up a server on your own computer. If you're a Mac user, I highly recommend **MAMP**, which has both a free and a premium paid version. You can get more information and download MAMP from `http://www.mamp.info`. If you're on Windows, I highly recommend **WampServer**, which is free. You can get more information and download WampServer from `http://www.wampserver.com`.

Time for action – building custom Ajax tooltips

Follow these steps to set up some tooltips that display Ajax content:

1. We'll get started by creating an HTML document and associated files and folders like we did in *Chapter 1*, *Designer, Meet jQuery*. Our HTML page should contain a couple paragraphs of text that have some links to further information. My first HTML document looks like the following:

```
<!DOCTYPE html>
<html>
<head>
   <title>Pittsburgh, Pennsylvania</title>
   <link rel="stylesheet" href="styles/styles.css"/>
```

```
</head>
<body>

<h2>Pittsburgh, Pennsylvania</h2>

<p>Pittsburgh is the second-largest city in the US Commonwealth of
Pennsylvania and the county seat of Allegheny County. Regionally,
it anchors the largest urban area of Appalachia and the Ohio
River Valley, and nationally, it is the 22nd-largest urban area
in the United States. The population of the city in 2010 was
305,704 while that of the seven-county metropolitan area stood at
2,356,285. <a href="infoboxes/downtown.html">Downtown Pittsburgh</
a> retains substantial economic influence, ranking at 25th in the
nation for jobs within the urban core and 6th in job density.</p>

<p>The characteristic shape of Pittsburgh's central business
district is a triangular tract carved by the confluence of the
Allegheny and Monongahela rivers, which form the Ohio River. The
city features 151 high-rise buildings, 446 bridges, two inclined
railways, and a pre-revolutionary fortification. Pittsburgh is
known colloquially as "The City of Bridges" and "The Steel City"
for its <a href="infoboxes/bridges.html">many bridges</a> and
former steel manufacturing base.</p>

<p>The warmest month of the year in Pittsburgh is July, with a
24-hour average of 72.6&deg;F. Conditions are often humid, and
combined with the 90&deg;F (occurring on an average of 8.4 days
per annum), a considerable <a href="infoboxes/heatindex.html">heat
index</a> arises.</p>

<script src="scripts/jquery.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

2. We need an easy way to select the three more information links, so we'll add a CSS class to each one like this:

```
<a href="infoboxes/downtown.html" class="infobox">Downtown
Pittsburgh</a>
```

3. Next, we need to create a set of short pages that each contain a photo and a caption for each of the links in my previous text. Here's a sample of one of my short HTML pages:

```
<!DOCTYPE html>
<html>
<head>
   <title>Downtown Pittsburgh</title>
</head>
```

```
<body>
   <img src="../images/downtown.jpg"/>
   <p>Downtown Pittsburgh</p>
</body>
</html>
```

As you can see, the file is extremely small and simple.

4. Create an `infoboxes` directory alongside the main page. Save your simple HTML file to this directory, and then create more simple files—one for each link in the main document.

5. Now, if you open the main page in a browser and click the links in the text, you'll see that these short, plain pages load up in the browser. We've got the basic functionality down, so next we'll move on to progressively enhancing our page for those with JavaScript enabled.

6. We'll use the purple color scheme that we set up earlier in the chapter for our tooltips, so let's add the CSS for the `ui-tooltip-purple` class to the `styles.css` file:

```
/*! Purple tooltip style */
.ui-tooltip-purple .ui-tooltip-titlebar,
.ui-tooltip-purple .ui-tooltip-content{
   border-color: #c1c3e6;
   color: #545aba;
}
   .ui-tooltip-purple .ui-tooltip-content{
      background-color: #f1f2fa;
   }
   .ui-tooltip-purple .ui-tooltip-titlebar{
      background-color: #d9daf0;
   }
   .ui-tooltip-purple .ui-state-default .ui-tooltip-icon{
      background-position: -102px 0;
   }
   .ui-tooltip-purple .ui-tooltip-icon{
      border-color: #c1c3e6;
   }
   .ui-tooltip-purple .ui-tooltip-titlebar .ui-state-hover{
   border-color: #c1c3e6;
   }
```

7. Now that we've got our HTML and CSS all set up, let's dive into the JavaScript. Attach the qTip plugin at the bottom of the page, between jQuery and your `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.qtip.min.js"></script>
<script src="scripts/scripts/js"></script>
</body>
</html>
```

8. Next, open `scripts.js` and we'll get started with our document ready function:

```
   $(document).ready(function(){
   });
```

9. Next, we're going to call the `qtip()` method in a slightly different way than we have before. Inside the `qtip()` method, we need to easily get to the information about just the link we're working with, so we're going to use jQuery's `each()` method to loop through them one at a time. That will look like this:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip()
   });
});
```

10. Now, if you refresh the page in your browser, you'll see that nothing happens when you hover over the links. This is because our links don't have `title` attributes, and that's what the qTip plugin is looking for by default. However, we can override that default to insert any content we'd like into our tooltips.

11. We're going to be displaying those simple HTML pages we set up inside our tooltips. Even though Ajax requests tend to be quick, there could still be a bit of a delay, so let's get ready to use Ajax by adding a loading message that will display for our site visitors while they wait for the real content to show up:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...'
         }
      });
   })
});
```

Now when you refresh the page in the browser, you'll see the tooltips contain the **Loading...** text.

12. We want to switch the behavior of the tooltips so that they show up when the link is clicked instead of when the mouse hovers over. We also want to make sure that only one tooltip is visible on the page at a time. If the site visitor opens a tooltip while another is already open, the first one should close so they don't end up with many tooltips open all over the screen. This is how we'll do that:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...'
         },
         show: {
            event: 'click',
            solo: true
         },
      });
   })
});
```

13. Now if you refresh the page in a browser, you'll see that the tooltip no longer appears when we hover over the links.

14. However, when we click on the links right now, we're taken to the short simple HTML page we set up. We have to tell the browser to ignore the link because we have other plans in mind. We can cancel the default behavior by adding this line of code above our earlier code and inside the document ready statement:

```
$(document).ready(function(){
$('a.infobox').bind('click', function(e){e.preventDefault()});
   $('a.infobox').each(function(){
```

15. What we're doing here is binding a function that fires when the links are clicked. Our function is pretty simple. We pass the current link to the function (e in this case for brevity, but we could have named it almost anything), and then we tell the browser to prevent the default link behavior.

    Now if you refresh the page in the browser, you'll see that the tooltips appear when we click on the links—clicking the links no longer takes us off to a new page.

16. But we could write our code in a more succinct way. Remember that jQuery allows us to chain methods, one right after the other. In this case, we can chain the `bind()` method directly to the end of the `each()` method we wrote earlier. The new JavaScript will look like this:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...'
         },
         show: {
            event: 'click',
            solo: true
         },
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

17. Next, let's adjust the style of our tooltips by adding a drop shadow and applying the purple color scheme we wrote to our tooltips:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...',
         },
         show: {
            event: 'click',
            solo: true
         },
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault();});
});
```

Now when you refresh the page in the browser, you'll see that we have purple tooltips that have a drop shadow. We're getting closer.

18. Next, let's add in the Ajax magic to load our simple HTML pages into the tooltips. Remember, this will only work from a server, so to see this step in action, you'll either have to upload your files to a server, or else set up a server on your own computer.

To tell the tooltips to fetch content via Ajax, all we have to do is pass the URL of the content we'd like to fetch. In this case, we've already linked out to that content. We just have to grab the link URL from each link. That's easily accessible to us by using the `attr()` method of jQuery. That will look like this:

```
$(this).attr('href')
```

In this case, `$(this)` is referring to the current link. I call the `attr()` method and pass that method the attribute I would like to fetch, in this case the `href` attribute of the link contains the information that I want. The `attr()` method can be used to fetch any attribute—an `src` attribute of an image, a `title` attribute of any element, a `cellspacing` attribute of a table, and so on:

```
$('img').attr('src')
$('p').attr('title')
$('table').attr('cellspacing')
```

19. Now that we know how to get the `href` attribute of our link, we'll use that to tell the tooltip which URL to use to get the content for our tooltip:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
         text: 'Loading...',
         ajax: {
             url: $(this).attr('href')
             }
         },
         show: {
             event: 'click',
             solo: true
         },
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

20. Refresh your browser and click on one of the links—you'll see the purple tooltip pop up with the HTML content from our simple HTML pages. Pretty amazing that fetching content with Ajax can be that simple, isn't it?

    Now, let's make a couple of other final tweaks to the tooltips to make them even better.

21. First, we'll add a title bar to the tooltips. To get some custom text for this, let's go back to each of the links in the `index.html` file and add a `title` attribute that contains the text to display at the top of the tooltips:

```
<a href="infoboxes/downtown.html" class="infobox" title="Downtown
Pittsburgh">Downtown Pittsburgh</a>
...
<a href="infoboxes/bridges.html" class="infobox" title="Pittsburgh
Bridges">many bridges</a>

<a href="infoboxes/heatindex.html" class="infobox" title="Beating
the Heat">heat index</a>
```

22. Now, we can fetch the `title` attribute of these links in much the same way that we fetched the URL of the `href` attribute and pass it to qTip as the title text for the tooltip. While we're at it, we can also pass in a `true` value for button to show a small close button at the top-right of the tooltip:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...',
            ajax: {
               url: $(this).attr('href')
            },
            title: {
               text: $(this).attr('title'),
               button: true
            }
         },
         show: {
            event: 'click',
            solo: true
         },
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

Now when you refresh the browser, you'll see a darker title bar with a close button appear at the top of each tooltip.

23. However, if you try to move your mouse over to click the close button, you'll see that the tooltip disappears before you can get there. We changed the show value of the tooltip to show on a click instead of on a mouse hover, but we never changed the hide value—the tooltip is still being hidden when we move our mouse off the link. This is a little bit awkward, so I'm going to change the hide value to `unfocus` so that the tooltip will be hidden when the link loses focus or when the site visitor clicks the close button on the tooltip:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...',
            ajax: {
               url: $(this).attr('href')
            },
            title: {
               text: $(this).attr('title'),
               button: true
            }
         },
         show: {
            event: 'click',
            solo: true
         },
         hide: 'unfocus',
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

24. Refresh your browser and you'll see that the interaction is much better now. Our site visitor doesn't have to carefully leave their mouse over the link in order to view the content inside our tooltip. And our tooltip is still easy to remove—the site visitor can click the close button, or click anywhere outside the tooltip on the page and the tooltips hide.

25. Now, there's just one thing left to do, and that's to position the tooltips right where we'd like them to appear. I want to show my tooltips centered below the links, so I'll match up the top-center of the tooltip with the bottom-center of the link:

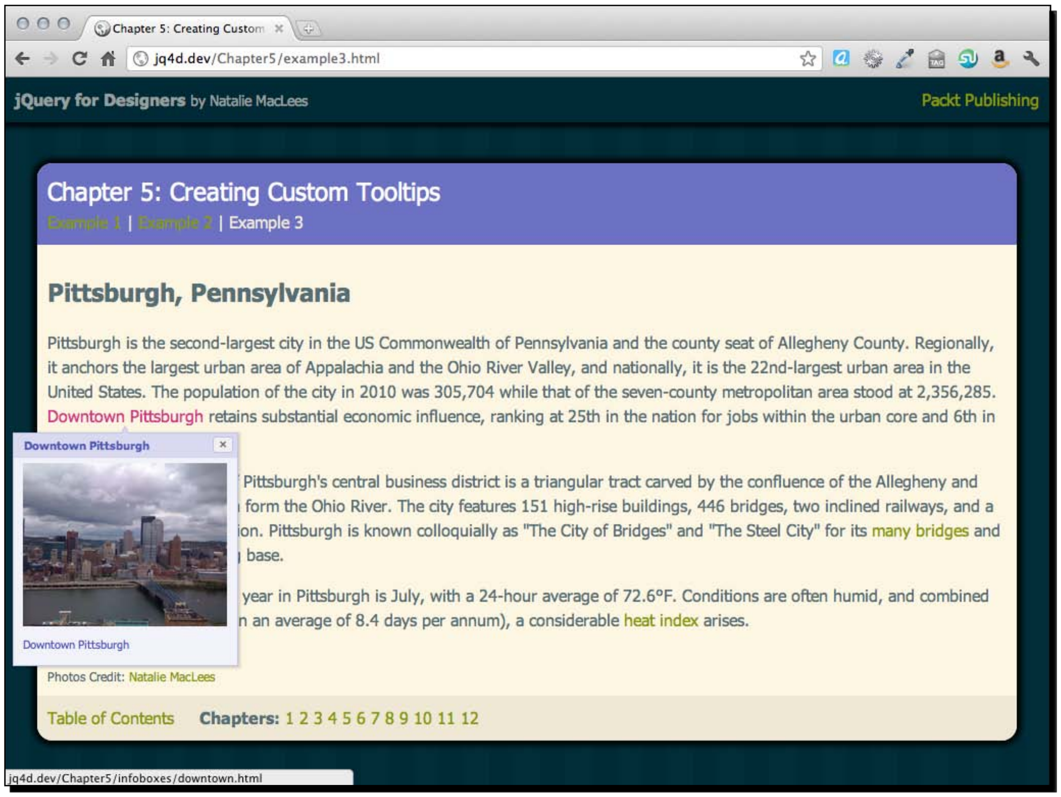
```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...',
            ajax: {
               url: $(this).attr('href')
            },
            title: {
               text: $(this).attr('title'),
               button: true
            }
         },
         position: {
            my: 'top center',
            at: 'bottom center'
         },
         show: {
            event: 'click',
            solo: true
         },
         hide: 'unfocus',
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

Now, if you refresh the page in the browser and click the links, you'll see the tooltips slide into place from their default position.

26. Our tooltips are looking good, but we still have a couple of problems. One is that the animation of the tooltip from the bottom corner to the middle of the tooltip is a little bit distracting. To work around this, let's set the `effect` value to `false`. That way the tooltip will show up where it's supposed to without the animation of sliding into place. The other problem is that, depending on the size of your browser window, sometimes the tooltips are cut off and display outside the screen area. To make sure this doesn't happen, we'll set the `viewport` value to the window like the following:

```
$(document).ready(function(){
   $('a.infobox').each(function(){
      $(this).qtip({
         content: {
            text: 'Loading...',
            ajax: {
               url: $(this).attr('href')
            },
            title: {
               text: $(this).attr('title'),
               button: true
            }
         },
         position: {
            my: 'top center',
            at: 'bottom center',
            effect: false,
            viewport: $(window)
         },
         show: {
            event: 'click',
            solo: true
         },
         hide: 'unfocus',
         style: {
            classes: 'ui-tooltip-purple ui-tooltip-shadow'
         }
      });
   }).bind('click', function(e){e.preventDefault()});
});
```

27. Now you'll see when you reload the page in the browser, that the tooltip will display centered below the link if possible, but if that would put it outside the window area, then the tooltip will adjust its position to the best possible place for display in relation to the link. We lose a bit of control over just where the tooltip appears, but we can make sure that our site visitors will always be able to see the tooltip's content, which is more important.


Chapter 5: Creating Custom
jq4d.dev/Chapter5/example3.html
jQuery for Designers by Natalie MacLees
Packt Publishing
jq4d.dev/Chapter5/infoboxes/downtown.html

Chapter 5: Creating Custom Tooltips

Example 1 | Example 2 | Example 3

Pittsburgh, Pennsylvania

Pittsburgh is the second-largest city in the US Commonwealth of Pennsylvania and the county seat of Allegheny County. Regionally, it anchors the largest urban area of Appalachia and the Ohio River Valley, and nationally, it is the 22nd-largest urban area in the United States. The population of the city in 2010 was 305,704 while that of the seven-county metropolitan area stood at 2,356,285. Downtown Pittsburgh retains substantial economic influence, ranking at 25th in the nation for jobs within the urban core and 6th in

Downtown Pittsburgh ×

Downtown Pittsburgh

Pittsburgh's central business district is a triangular tract carved by the confluence of the Allegheny and form the Ohio River. The city features 151 high-rise buildings, 446 bridges, two inclined railways, and a ion. Pittsburgh is known colloquially as "The City of Bridges" and "The Steel City" for its many bridges and base.

year in Pittsburgh is July, with a 24-hour average of 72.6°F. Conditions are often humid, and combined n an average of 8.4 days per annum), a considerable heat index arises.

Photos Credit: Natalie MacLees

Table of Contents Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

Summary

We covered a lot of ground in this chapter. We learned how to use the qTip plugin to replace the browser's default tooltips with custom-designed tooltips. We saw how to take the customization a bit further by adding speech-bubble tooltips to a navigation bar. And finally, we used Ajax to pull in some external content, customizing not only the appearance of the tooltip, but also pulling in custom content, adding a title bar and close button, ensuring the tooltip would always be visible, and customizing the show and hide behaviors of the tooltip. I hope that you can see how flexible the qTip plugin is and how many uses it can have beyond just customizing the appearance of tooltips. Have fun experimenting with all the different settings listed in the plugin's documentation and see how creative you can be in customizing the appearance of your tooltips.

Next up, we'll take a look at creating nicely designed and animated dropdown navigation menus.

6 Building an Interactive Navigation Menu

In 2003, an article published on A List Apart (`http://alistapart.com`) *called Suckerfish Dropdowns showed how HTML and CSS alone (with just a little JavaScript help for IE6) could be used to build a complex multilevel drop-down menu. The Suckerfish name derived from the gorgeously designed demo of the technique which featured illustrations of remoras and sharksuckers. While useful, the original requires that the site visitor not move their mouse outside the menu area while navigating or the menu disappears. Over the years, the Suckerfish Dropdowns have inspired a lot of spinoffs—Son of Suckerfish, Improved Suckerfish, and so on—that attempt to address the shortcomings of the original. Since jQuery can make everything better, we'll build on this idea using the Superfish jQuery plugin to make the menu easier to use.*

The developer of the Superfish plugin, Joel Birch, says that most support issues with the plugin come from people not understanding the CSS for the menu. To be sure you have a firm grasp on the CSS, I highly recommend reading the original Suckerfish Dropdowns article on *A List Apart* at `http://www.alistapart.com/articles/dropdowns`.

To get started with this plugin, we'll build on a basic Suckerfish menu. Since that menu only requires CSS, we still get an interactive menu if we have JavaScript disabled. The menu is just improved for users with JavaScript enabled.

In this chapter, we'll learn the following topics:

- Using the Superfish jQuery plugin to create a horizontal drop-down menu
- Creating vertical flyout menu with the Superfish plugin
- Customizing the drop-down and flyout menus created with the Superfish plugin

Horizontal drop-down menu

The horizontal drop-down menu was long a common item in desktop software but challenging if not impossible to implement in websites until CSS and JavaScript finally arrived on the scene to make them possible.

Time for action – creating a horizontal drop-down menu

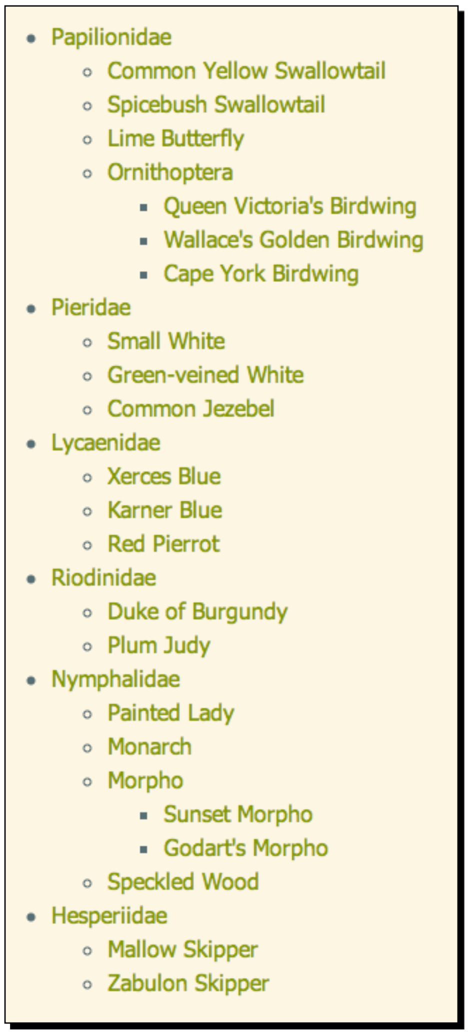
Let's take a look at how we can use the Superfish plugin to create a horizontal drop-down menu:

1. To get started, we'll create a simple HTML page and the associated folders and files like we created in *Chapter 1*, *Designer, Meet jQuery*. The body of our HTML file will contain a navigation menu that consists of nested unordered lists as follows:

```
<ul id="sfNav" class="sf-menu">
  <li><a href="#">Papilionidae</a>
    <ul>
      <li><a href="#">Common Yellow Swallowtail</a></li>
      <li><a href="#">Spicebush Swallowtail</a></li>
      <li><a href="#">Lime Butterfly</a></li>
      <li><a href="#">Ornithoptera</a>
        <ul>
          <li><a href="#">Queen Victoria's Birdwing</a></li>
          <li><a href="#">Wallace's Golden Birdwing</a></li>
          <li><a href="#">Cape York Birdwing</a></li>
        </ul>
      </li>
    </ul>
  </li>
  <li><a href="#">Pieridae</a>
    <ul>
      <li><a href="#">Small White</a></li>
      <li><a href="#">Green-veined White</a></li>
      <li><a href="#">Common Jezebel</a></li>
    </ul>
  </li>
  <li><a href="#">Lycaenidae</a>
    <ul>
      <li><a href="#">Xerces Blue</a></li>
      <li><a href="#">Karner Blue</a></li>
      <li><a href="#">Red Pierrot</a></li>
    </ul>
  </li>
  <li><a href="#">Riodinidae</a>
    <ul>
      <li><a href="#">Duke of Burgundy</a></li>
      <li><a href="#">Plum Judy</a></li>
    </ul>
  </li>
  <li><a href="#">Nymphalidae</a>
    <ul>
      <li><a href="#">Painted Lady</a></li>
      <li><a href="#">Monarch</a></li>
      <li><a href="#">Morpho</a>
        <ul>
          <li><a href="#">Sunset Morpho</a></li>
          <li><a href="#">Godart's Morpho</a></li>
        </ul>
      </li>
      <li><a href="#">Speckled Wood</a></li>
    </ul>
  </li>
  <li><a href="#">Hesperiidae</a>
    <ul>
      <li><a href="#">Mallow Skipper</a></li>
      <li><a href="#">Zabulon Skipper</a></li>
    </ul>
  </li>
</ul>
```

Note that we've added an `id` of `sfNav` and a `class` of `sf-menu` to the `<ul>` that contains our menu. This will make it easy for us to select and style the menu the way we'd like.

If you view your page in the browser, it will look similar to the following screenshot:



- Papilionidae
  - Common Yellow Swallowtail
  - Spicebush Swallowtail
  - Lime Butterfly
  - Ornithoptera
    - Queen Victoria's Birdwing
    - Wallace's Golden Birdwing
    - Cape York Birdwing
- Pieridae
  - Small White
  - Green-veined White
  - Common Jezebel
- Lycaenidae
  - Xerces Blue
  - Karner Blue
  - Red Pierrot
- Riodinidae
  - Duke of Burgundy
  - Plum Judy
- Nymphalidae
  - Painted Lady
  - Monarch
  - Morpho
    - Sunset Morpho
    - Godart's Morpho
  - Speckled Wood
- Hesperiidae
  - Mallow Skipper
  - Zabulon Skipper

As you can see, we've organized our links into a hierarchy. This is useful for finding the information that we want, but it takes up quite a lot of space. This is where we can use a technique of hiding extra information until it's needed.

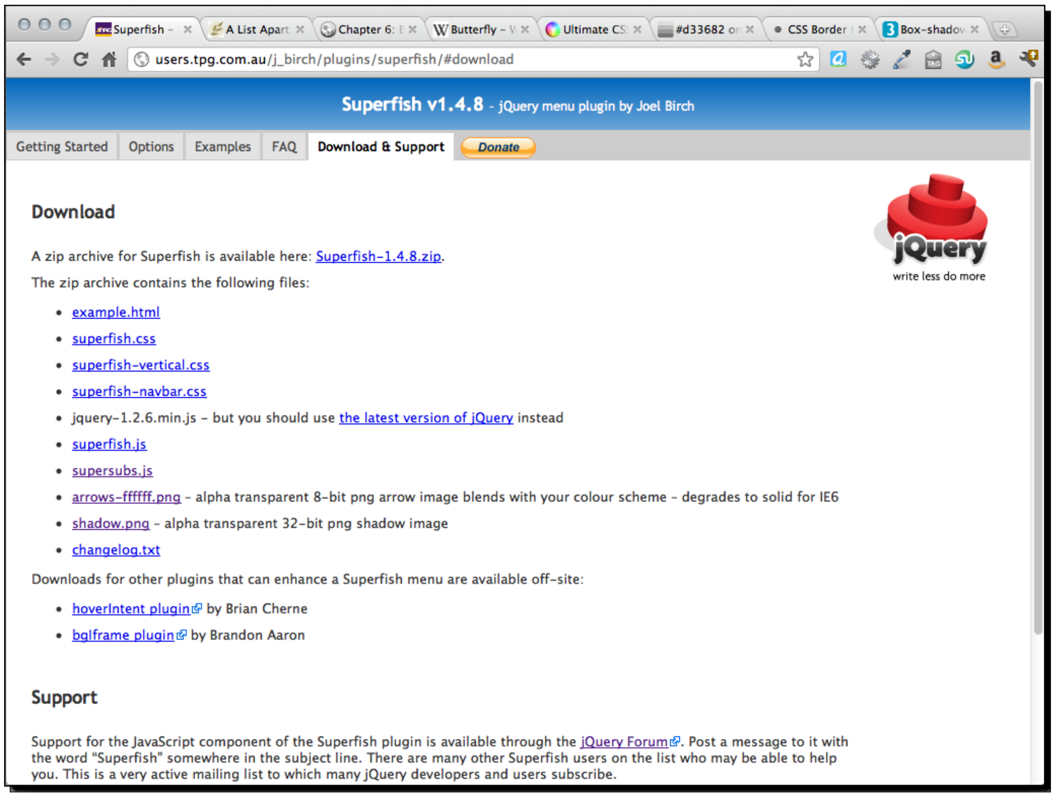
2. Next, we need a copy of the Superfish plugin. Head over to `http://users.tpg.com.au/j_birch/plugins/superfish/` where you'll find Joel Birch's Superfish plugin available for download along with documentation and examples.

In Joel's **Quick Start Guide**, we see that there are three simple steps to implementing the Superfish plugin:

- Write the CSS to create a Suckerfish-style drop-down menu
- Link to the `superfish.js` file
- Call the `superfish()` method on the element that contains your menu

Lucky for us, Joel also includes a sample CSS file, so we can get started quickly. We'll look at customizing the appearance of our menu later, but for now, we'll go ahead and use the CSS included with the plugin.

3. Click on the **Download & Support** tab.



users.tpg.com.au/j_birch/plugins/superfish/#download
Superfish v1.4.8 - jQuery menu plugin by Joel Birch
Getting Started
Options
Examples
FAQ
Download & Support
Donate
jQuery
write less do more

Download

A zip archive for Superfish is available here: Superfish-1.4.8.zip.

The zip archive contains the following files:

- example.html
- superfish.css
- superfish-vertical.css
- superfish-navbar.css
- jquery-1.2.6.min.js – but you should use the latest version of jQuery instead
- superfish.js
- supersubs.js
- arrows-ffffff.png – alpha transparent 8-bit png arrow image blends with your colour scheme – degrades to solid for IE6
- shadow.png – alpha transparent 32-bit png shadow image
- changelog.txt

Downloads for other plugins that can enhance a Superfish menu are available off-site:

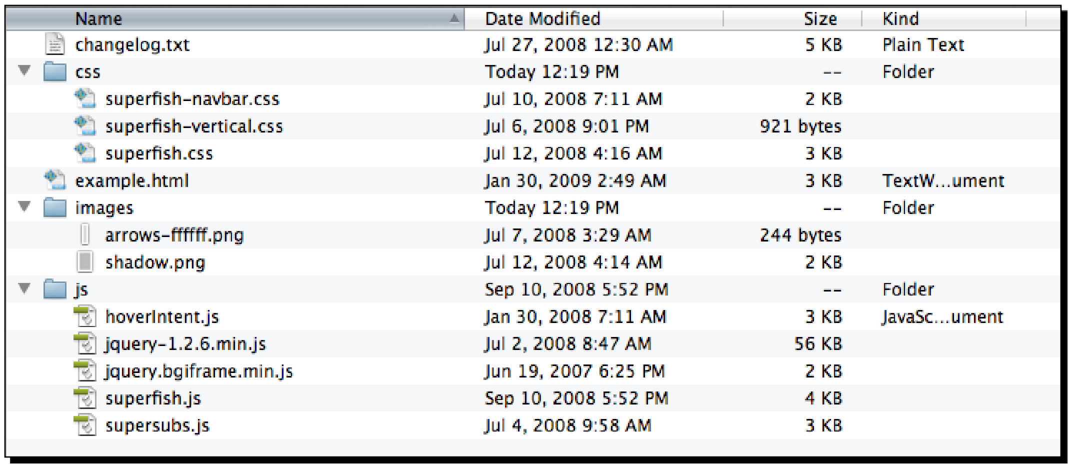
- hoverIntent plugin by Brian Cherne
- bgIframe plugin by Brandon Aaron

Support

Support for the JavaScript component of the Superfish plugin is available through the jQuery Forum. Post a message to it with the word "Superfish" somewhere in the subject line. There are many other Superfish users on the list who may be able to help you. This is a very active mailing list to which many jQuery developers and users subscribe.

The first link in the **Download** section is the link to download the ZIP file. Underneath that, we see a bulleted list of all the files included in the ZIP and links are provided to download each of them separately. We'll go ahead and download the entire ZIP file since we're going to make use of several of these files. Click on the **Superfish-1.4.8.zip** link and save the file to your computer.

4. Unzip the folder and take a look inside:



Name	Date Modified	Size	Kind
changelog.txt	Jul 27, 2008 12:30 AM	5 KB	Plain Text
▼ css	Today 12:19 PM	--	Folder
superfish-navbar.css	Jul 10, 2008 7:11 AM	2 KB	
superfish-vertical.css	Jul 6, 2008 9:01 PM	921 bytes	
superfish.css	Jul 12, 2008 4:16 AM	3 KB	
example.html	Jan 30, 2009 2:49 AM	3 KB	TextW...ument
▼ images	Today 12:19 PM	--	Folder
arrows-ffffff.png	Jul 7, 2008 3:29 AM	244 bytes	
shadow.png	Jul 12, 2008 4:14 AM	2 KB	
▼ js	Sep 10, 2008 5:52 PM	--	Folder
hoverIntent.js	Jan 30, 2008 7:11 AM	3 KB	JavaSc...ument
jquery-1.2.6.min.js	Jul 2, 2008 8:47 AM	56 KB	
jquery.bgiframe.min.js	Jun 19, 2007 6:25 PM	2 KB	
superfish.js	Sep 10, 2008 5:52 PM	4 KB	
supersubs.js	Jul 4, 2008 9:58 AM	3 KB	

We'll find the files nicely organized into subdirectories by type along with an example HTML file we can examine to see the plugin at work.

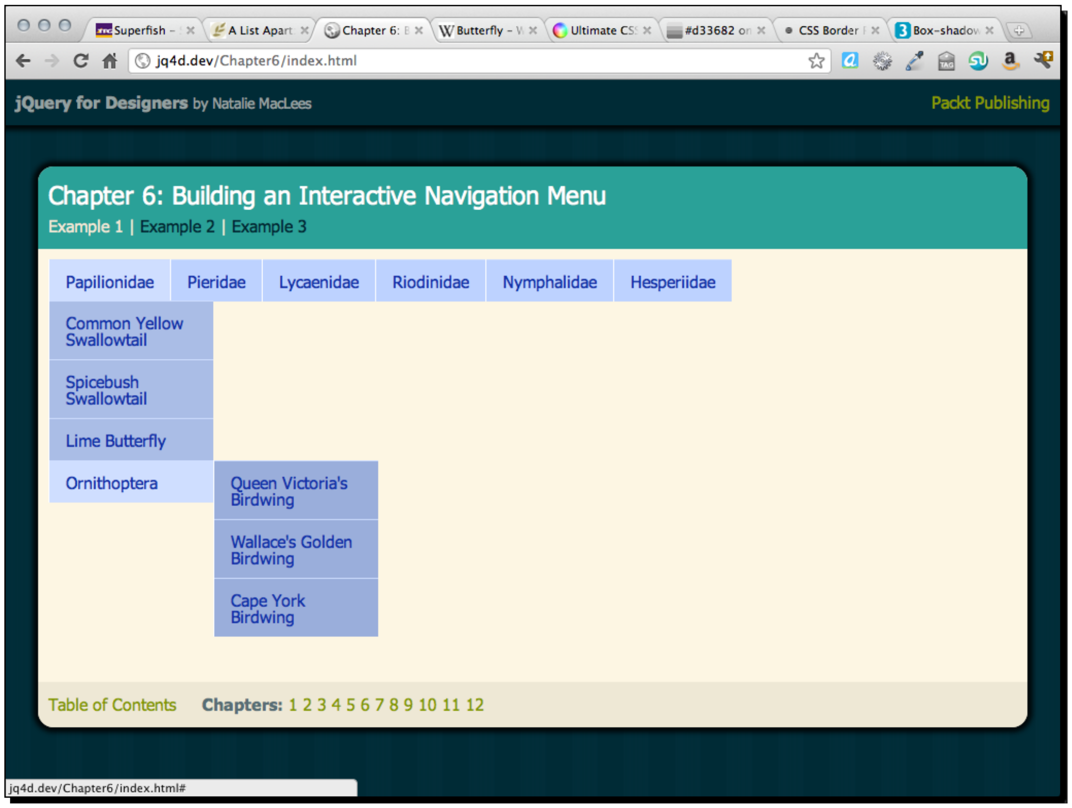
5. The first file we'll need from the **Download** section is the `superfish.css` file from the `css` folder. Copy that file to your own `styles` folder.

6. Next, we'll edit our HTML file to include the `superfish.css` file in the head of the document:

```
<head>
  <title>Chapter 6: Building an Interactive Navigation Menu
</title>
  <link rel="stylesheet" href="styles/superfish.css"/>
  <link rel="stylesheet" href="styles/styles.css"/>
</head>
```

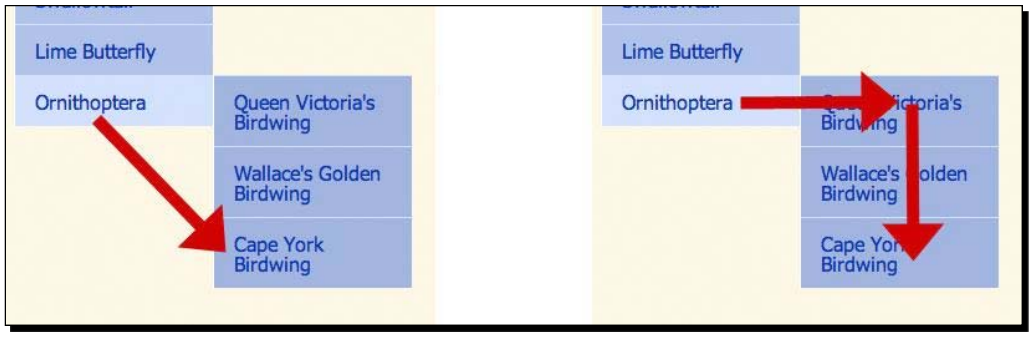
We're attaching the `superfish.css` file before our `styles.css` file to make it easier for us to override any styles in the `superfish.css` file we want to change later.

7. Now, if you refresh the page in a browser, you'll see a working Suckerfish drop-down menu:


jq4d.dev/Chapter6/index.html
jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 6: Building an Interactive Navigation Menu
Example 1 | Example 2 | Example 3
Papilionidae
Pieridae
Lycaenidae
Riodinidae
Nymphalidae
Hesperiidae
Common Yellow Swallowtail
Spicebush Swallowtail
Lime Butterfly
Ornithoptera
Queen Victoria's Birdwing
Wallace's Golden Birdwing
Cape York Birdwing
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12
jq4d.dev/Chapter6/index.html#

When I move my mouse over the first link, the nested `<ul>` becomes visible. If I move my mouse down to the last link in the drop down, the `<ul>` nested at the third level becomes visible.

Keep in mind, all of this is accomplished without JavaScript— just CSS. If you spend a few moments using the menu, you'll probably quickly recognize some shortcomings. First, if I want to move my mouse from the **Ornithoptera** link to the **Cape York Birdwing** link, my natural inclination is to move my mouse diagonally. However, as soon as my mouse leaves the blue menu area, the menu closes and disappears. I have to adjust to move my mouse directly right onto the submenu, then down to the link I'm interested in.



This is awkward and makes the menu feel fragile. If my mouse moves even 1 pixel outside the menu, the menu collapses and disappears. Another problem is that the menu opens as soon as the mouse hovers over it. If I am moving my mouse over the menu moving from one part of the page to another, the menu opens and closes quickly, which can be distracting and unexpected.

This is a great place for jQuery to step in and make things a bit better and more usable.

Time for action – improving the drop-down menu with jQuery

Follow these steps to improve the usability of the drop-down menu with jQuery:

1. We'll begin by attaching the Superfish plugin to our HTML page at the bottom of our file, between jQuery and our `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/superfish.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

2. Next, open `scripts.js`, where we will write our code calling the `superfish()` method. As usual, we'll get started with the document ready statement so that our script runs as soon as the page is loaded into the browser:

```
  $(document).ready(function(){
    // Our code will go here.
  });
```

3. Looking at the documentation for the Superfish plugin, we see that we only have to select the element or elements that we'd like to apply the behavior to and then call the `superfish()` method. Inside our `ready()` method, we'll add the following code:

```
$(document).ready(function(){
  $('#sfNav').superfish();
});
```

Now, if you refresh the page in the browser, you'll see the menu still looking very similar, but with much improved behavior. The Superfish JavaScript and CSS work together to add arrows to the menu items that have nested children. If you move your mouse off of the menu, it does not disappear immediately, making it possible to move the mouse diagonally to nested menu items. There's also a subtle fade in animation when the menu items appear. And a background color change to each menu item on hover, making it easy to see which item is currently active.

What just happened?

We set up a navigation menu consisting of a set of nested lists, forming a hierarchy. Next, we attached a CSS file to add a simple drop-down functionality to our menu. However, that CSS-only menu had a few shortcomings. So we attached the Superfish plugin to take care of those and make our menu more user friendly.

Vertical fly-out menu

We saw how the addition of the Superfish plugin enhanced the user experience of our drop-down menu, but what if we wanted to create a vertical fly-out menu instead?

Time for action – creating a vertical fly-out menu

Switching from a horizontal drop-down menu to a vertical fly-out menu couldn't be easier. We'll use the same HTML markup and our JavaScript code will stay the same. The only change we'll need to make is to add some new CSS to make our menu display vertically instead of horizontally. We can keep working with the same files we used in the last example.

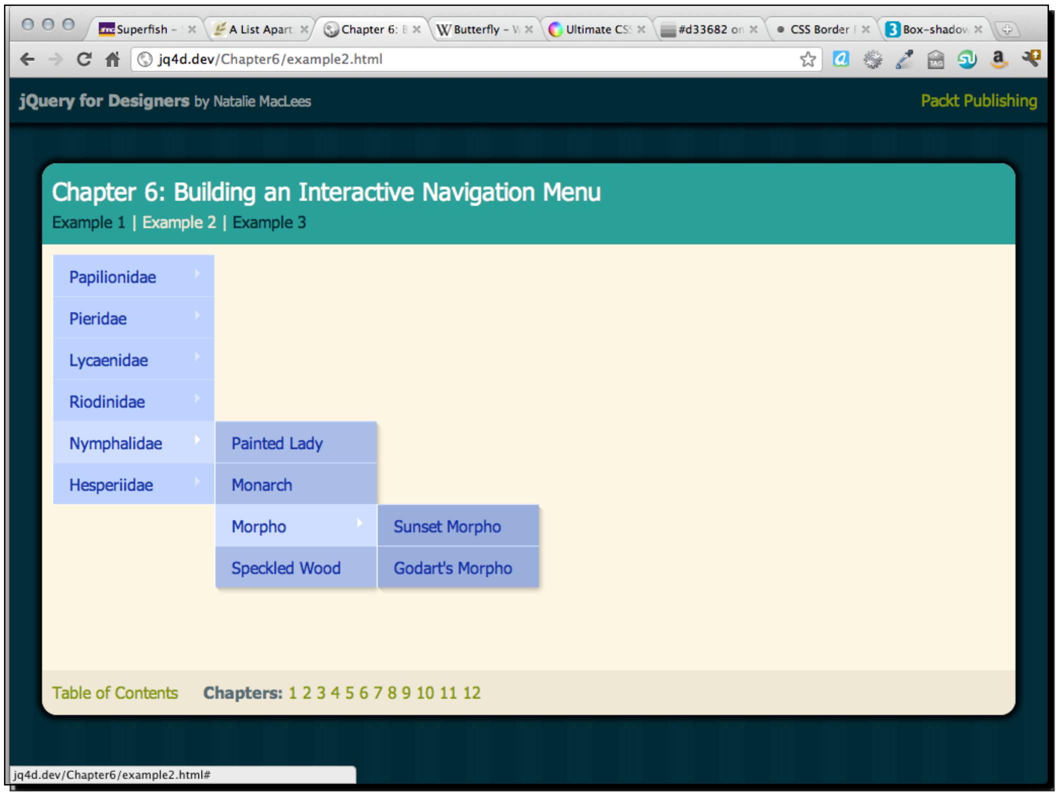
1. In the `css` folder of the Superfish download, you'll find a file named `superfish-vertical.css`. Copy that file to your own `styles` folder. In the `head` section of the HTML file, we'll attach the new CSS file. Between `superfish.css` and `styles.css`, add the new CSS file:

```
<link rel="stylesheet" href="styles/superfish.css"/>
  <link rel="stylesheet" href="styles/superfish-vertical.css"/>
<link rel="stylesheet" href="styles/styles.css"/>
```

2. Now, in the HTML we'll add an `sf-vertical` class to the list that contains our menu.

   ```
   <ul id="sfNav" class="sf-menu sf-vertical">
   ```

3. Now when you refresh the page in a browser, you'll see the menu displayed vertically with fly-outs:


jq4d.dev/Chapter6/example2.html
jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 6: Building an Interactive Navigation Menu
Example 1 | Example 2 | Example 3
Papilionidae
Pieridae
Lycaenidae
Riodinidae
Nymphalidae
Hesperiidae
Painted Lady
Monarch
Morpho
Speckled Wood
Sunset Morpho
Godart's Morpho
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12
jq4d.dev/Chapter6/example2.html#

What just happened?

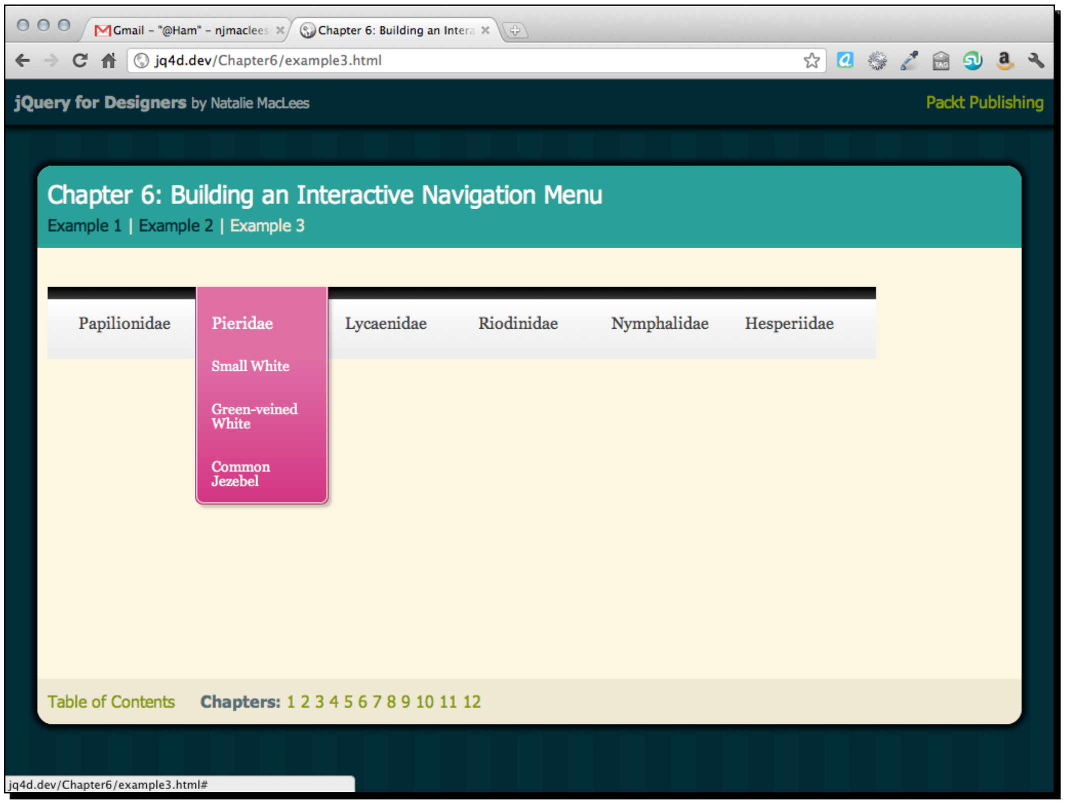
The only difference between the horizontal drop-down menu and the vertical fly-out menu is the CSS and a class name added to the menu container. By simply adding a new CSS file and a new CSS class, it's possible to create a vertical fly-out menu instead of a horizontal drop-down menu.

Customizing the navigation menu

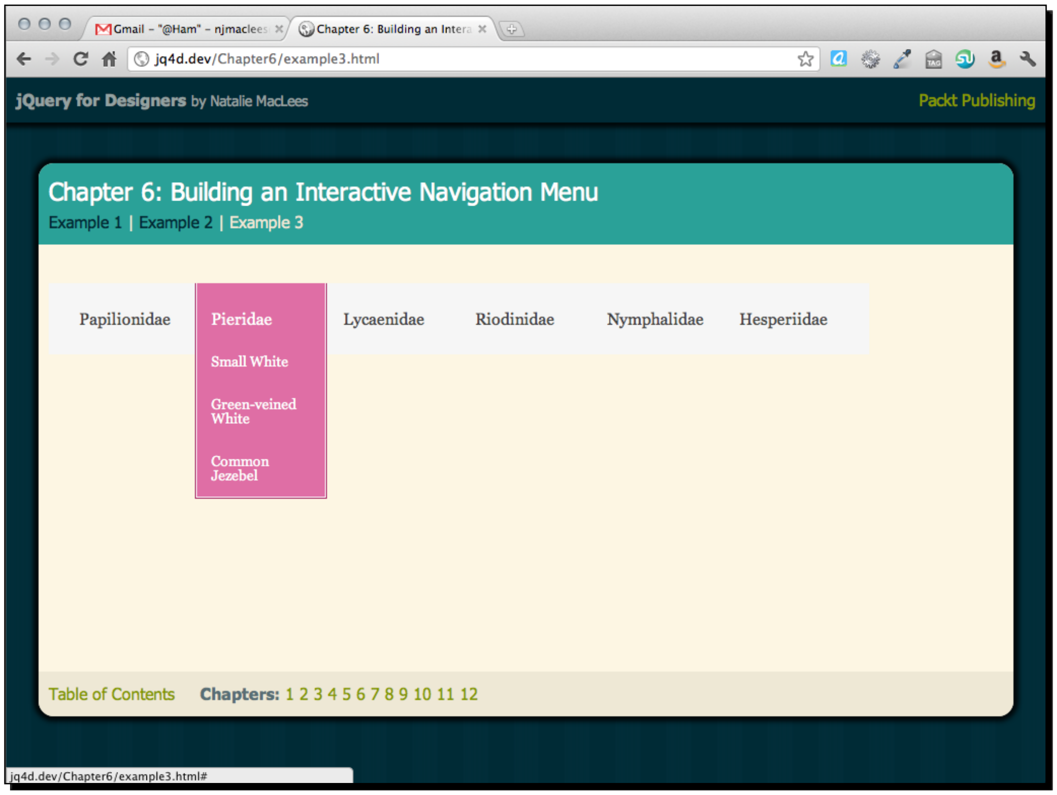
The included CSS with the Superfish plugin makes creating an interactive navigation menu quick and simple, but a soft blue menu isn't going to fit into every design, so let's take a look at how we can customize the menu.

We're going to take a look at how we can customize the look of the menu by writing our own CSS, customize the animation for showing the nested menus, highlight the current page, and enhance the hover behavior of the menu.

We're going to get started by writing some CSS to create a custom look for our menus. We're going to use the Suckerfish Dropdown approach to create a menu that will work for our site visitors who don't have JavaScript enabled. I'd like to create a soft gradient background and have my menu items appear to be ribbons floating over the top of this background. My menu will look similar to the following screenshot:


jq4d.dev/Chapter6/example3.html
jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 6: Building an Interactive Navigation Menu
Example 1 | Example 2 | Example 3
Papilionidae
Pieridae
Small White
Green-veined White
Common Jezebel
Lycaenidae
Riodinidae
Nymphalidae
Hesperiidae
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

I'm going to really take advantage of the newer CSS3 properties available for use in modern browsers. I'm using gradients, box shadows, and rounded corners. I've carefully selected these options because even without these extras, the menu will still look okay and will be usable. The following is an example of how the menu will appear in the older browsers:


Gmail - "@Ham" - njmaclees
Chapter 6: Building an Intera
jq4d.dev/Chapter6/example3.html
jQuery for Designers by Natalie MacLees
Packt Publishing
Chapter 6: Building an Interactive Navigation Menu
Example 1 | Example 2 | Example 3
Papilionidae
Pieridae
Small White
Green-veined White
Common Jezebel
Lycaenidae
Riodinidae
Nymphalidae
Hesperiidae
Table of Contents
Chapters: 1 2 3 4 5 6 7 8 9 10 11 12
jq4d.dev/Chapter6/example3.html#

You can see that it's missing some of the extra styling from the modern browser example, but that it's still perfectly usable and generally pleasing to the eye. If it were really important for the menu to look the same in all browsers, then we could apply the same effects using images instead of CSS3 to get the final effect. However, we'd likely need to add some extra markup, and we'd definitely need to add images and extra lines of CSS, all adding weight to our pages overall. Whether you decide to allow your menu to degrade gracefully for older browsers or you decide to write the extra code and create the extra images to make the menu appear the same in all browsers is a decision you'll need to make based on the client's expectations, the target audience for the website, and the importance of building speedy and lightweight pages.

Here are some things to keep in mind as you write custom CSS for a drop-down or fly-out menu:

:hover and .sfHover

In order to make your menu work without JavaScript, you'll need to take advantage of the `:hover` pseudo-class for the list items. Make sure to also create a CSS selector for the same element with a `.sfHover` class, which will be used by the JavaScript. For example:

```
.sf-menu li.sfHover ul,
.sf-menu li:hover ul  {
  left: -1px;
  top: 70px; /* match top ul list item height */
  z-index: 99;
  }
```

This bit of code makes the nested menu visible on the screen when the parent list item is hovered over. Including the `li:hover` selector ensures the menu works without JavaScript. Also including the `li.sfHover` selector ensures the JavaScript menu will apply the same code.

Cascading inherited styles

It's the very nature of CSS for styles to cascade down the DOM and be applied to all children of the selector as well as the selector itself. So, if you write code to style the list items of the first-level menu like this:

```
ul.sf-menu li  {
  background: #cc0000; /* Dark red background */
}
```

All of the `<li>`s in your menu are going to have a dark red background, no matter which level of the menu they appear in. If you want to apply different styles to different menu levels, you'll have to override the cascade in other lines of code. For example, if I wanted to make the second menu level have a dark blue background, I'd add this snippet of CSS *after* the preceding code:

```
ul.sf-menu li li  {
  background: #0000cc; /* Dark blue background */
}
```

That means for a `<li>` inside another `<li>`, the background will be blue. Keep in mind that now this style will in turn cascade down to other menu levels, so if you want a dark green background for the third-level menu, you'll need to add another bit of CSS:

```
ul.sf-menu li li li  {
  background: #00cc00; /* Dark green background */
}
```

In some cases, making use of direct descendent selectors in your CSS can help to prevent you from having to write too many lines of CSS overriding styles written for elements higher up in the DOM. For example:

```
ul.sf-menu > li  {
  background: #cc0000; /* Dark red background */
}
```

This bit of CSS takes advantage of a direct descendent selector (`>`). The dark red background in this case will only apply to the `<li>` elements nested directly inside `<ul>` with a class of `sf-menu`. It will not cascade down to the second- or third-level menus.

Vendor prefixes

If you'd like to experiment with the new CSS3 properties, you'll have to be sure to prepend your properties with vendor-specific prefixes. While these properties are supported by most modern browsers, they are still under development and may be implemented in slightly different ways in different browsers. Take for example, this bit of CSS, which rounds the bottom two corners:

```
.sf-menu ul li:last-child a  {
  -webkit-border-bottom-right-radius: 7px;
  -webkit-border-bottom-left-radius: 7px;
  -moz-border-radius-bottomright: 7px;
  -moz-border-radius-bottomleft: 7px;
  border-bottom-right-radius: 7px;
  border-bottom-left-radius: 7px;
  }
```

You can see that the property for the bottom-left and bottom-right corners is slightly different between webkit-based browsers (Safari and Chrome, mainly) and Mozilla browsers (mainly Firefox). After the vendor-specific code, include the general CSS3 code for any browsers that support that to make sure your code is future-proof.

Time for action – customizing Superfish menus

Customizing a Superfish menu mostly involves writing your own CSS to style the menu the way you'd like. Here's how we'll create a custom look for the menu:

If you'll remember some web basics, you'll remember that CSS stands for Cascading Style Sheets. The cascading features are what we'll focus on here. Any styles we write for the top level of our menu are going to cascade down to the other levels of the menu. We have to remember that and handle all the cases where we'd rather stop a style from cascading downward.

1. Let's get started by styling the top level of our menu. Since I'm using new CSS3 features, we're going to have to be prepared to write a bit of extra code so that each browser can handle our code gracefully. Here's the CSS we'll create for the top level of the menu. Place this code inside your `styles.css` file:

```
/**** Level 1 ****/
.sf-menu,
.sf-menu *  {
  list-style:  none;
  margin: 0;
  padding: 0;
  }

.sf-menu  {
  background: #f6f6f6; /* Old browsers */
  background: -moz-linear-gradient(top, rgba(0,0,0,1) 1%,
rgba(56,56,56,1) 16%, rgba(255,255,255,1) 17%, rgba(246,246,246,1)
47%, rgba(237,237,237,1) 100%); /* FF3.6+ */
  background: -webkit-gradient(linear, left top,
left bottom, color-stop(1%,rgba(0,0,0,1)), color-
stop(16%,rgba(56,56,56,1)), color-stop(17%,rgba(255,255,255,1)),
color-stop(47%,rgba(246,246,246,1)), color-stop(100%,rg
ba(237,237,237,1))); /* Chrome,Safari4+ */
  background: -webkit-linear-gradient(top, rgba(0,0,0,1)
1%,rgba(56,56,56,1) 16%,rgba(255,255,255,1)
17%,rgba(246,246,246,1) 47%,rgba(237,237,237,1) 100%); /*
Chrome10+,Safari5.1+ */
  background: -o-linear-gradient(top, rgba(0,0,0,1)
1%,rgba(56,56,56,1) 16%,rgba(255,255,255,1)
17%,rgba(246,246,246,1) 47%,rgba(237,237,237,1) 100%); /*
Opera11.10+ */
  background: -ms-linear-gradient(top, rgba(0,0,0,1)
1%,rgba(56,56,56,1) 16%,rgba(255,255,255,1)
17%,rgba(246,246,246,1) 47%,rgba(237,237,237,1) 100%); /* IE10+ */
```

```
  filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#000000', endColorstr='#ededed',GradientType=0 );
/* IE6-9 */
  background: linear-gradient(top, rgba(0,0,0,1)
1%,rgba(56,56,56,1) 16%,rgba(255,255,255,1)
17%,rgba(246,246,246,1) 47%,rgba(237,237,237,1) 100%); /* W3C */
  float: left;
  font-family: georgia, times, 'times new roman', serif;
  font-size: 16px;
  line-height: 14px;
  margin: 28px 0 14px 0;
  padding: 0 14px;
  }

.sf-menu li  {
  border-left: 1px solid transparent;
  border-right: 1px solid transparent;
  float: left;
  position: relative;
  }

.sf-menu li.sfHover,
.sf-menu li:hover  {
  visibility: inherit; /* fixes IE7 'sticky bug' */
  }

.sf-menu li.sfHover,
.sf-menu li:hover  {
  background: #DF6EA5;
  border-color: #a22361;
  -webkit-box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  -moz-box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  }

.sf-menu a  {
  border-left: 1px solid transparent;
  border-right: 1px solid transparent;
  color: #444;
  display: block;
  padding: 28px 14px;
  position: relative;
  width: 98px;
  text-decoration: none;
  }

.sf-menu li.sfHover a,
.sf-menu li:hover a  {
  background: #DF6EA5;
  border-color: #fff;
  color: #fff;
  outline: 0;
  }

.sf-menu a,
.sf-menu a:visited  {
  color: #444;
  }
```

Phew! That seems like a lot of code, but much of it is the repeated gradient and shadow declarations we have to use for each different type of browser. Keep your fingers crossed that this requirement goes away soon and the browser vendors eventually reach agreement on how gradients and drop shadows should be created with CSS.

2. Next, let's take a look at how we'll style the next level of our menus. Add the following CSS to your `styles.css` file to style the second level:

```
/***** Level 2 ****/
.sf-menu ul  {
  background: rgb(223,110,165); /* Old browsers */
  background: -moz-linear-gradient(top, rgba(223,110,165,1) 0%,
rgba(211,54,130,1) 100%); /* FF3.6+ */
  background: -webkit-gradient(linear, left top, left
bottom, color-stop(0%,rgba(223,110,165,1)), color-
stop(100%,rgba(211,54,130,1))); /* Chrome,Safari4+ */
  background: -webkit-linear-gradient(top, rgba(223,110,165,1)
0%,rgba(211,54,130,1) 100%); /* Chrome10+,Safari5.1+ */
  background: -o-linear-gradient(top, rgba(223,110,165,1)
0%,rgba(211,54,130,1) 100%); /* Opera11.10+ */
  background: -ms-linear-gradient(top, rgba(223,110,165,1)
0%,rgba(211,54,130,1) 100%); /* IE10+ */
  filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#df6ea5', endColorstr='#d33682',GradientType=0 );
/* IE6-9 */
  background: linear-gradient(top, rgba(223,110,165,1)
0%,rgba(211,54,130,1) 100%); /* W3C */
  -webkit-border-bottom-right-radius: 7px;
  -webkit-border-bottom-left-radius: 7px;
  -moz-border-radius-bottomright: 7px;
  -moz-border-radius-bottomleft: 7px;
  border-bottom-right-radius: 7px;
  border-bottom-left-radius: 7px;
  border: 1px solid #a22361;
  border-top: 0 none;
  margin: 0;
  padding: 0;
  position: absolute;
  top: -999em;
  left: 0;
  width: 128px;
  -webkit-box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  -moz-box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  box-shadow: 3px 3px 3px rgba(0,0,0,0.2);
  font-size: 14px;
  }

.sf-menu ul li  {
  border-left: 1px solid #fff;
  border-right: 1px solid #fff;
  display: block;
  float: none;
  }

.sf-menu ul li:last-child  {
  border-bottom: 1px solid #fff;
  -webkit-border-bottom-right-radius: 7px;
  -webkit-border-bottom-left-radius: 7px;
  -moz-border-radius-bottomright: 7px;
  -moz-border-radius-bottomleft: 7px;
  border-bottom-right-radius: 7px;
  border-bottom-left-radius: 7px;
  }

.sf-menu ul li:last-child a  {
  -webkit-border-bottom-right-radius: 7px;
  -webkit-border-bottom-left-radius: 7px;
  -moz-border-radius-bottomright: 7px;
  -moz-border-radius-bottomleft: 7px;
  border-bottom-right-radius: 7px;
  border-bottom-left-radius: 7px;
  }

.sf-menu li.sfHover li.sfHover,
.sf-menu li:hover li:hover  {
  -webkit-box-shadow: none;
  -moz-box-shadow: none;
  box-shadow: none;
  }

.sf-menu li.sfHover li.sfHover  {
  border-right-color: #cb2d79
  }

.sf-menu li li a  {
  border: 0 none;
  padding: 14px;
  }

.sf-menu li li:first-child a  {
  padding-top: 0;
  }

.sf-menu li li.sfHover a,
.sf-menu li li:hover a  {
  background: transparent;
  border: 0 none;
  color: #f8ddea;
  outline: 0;
  }

.sf-menu li li a:hover  {
  color: #f8ddea;
  }

.sf-menu li.sfHover li a,
.sf-menu li:hover li a  {
  background: transparent;
  }

.sf-menu li.sfHover li.sfHover a  {
  background: #cb2d79;
  }
```

```
.sf-menu li.sfHover ul,
.sf-menu li:hover ul  {
  left: -1px;
  top: 70px; /* match top ul list item height */
  z-index: 99;
  }

.sf-menu li li.sfHover,
.sf-menu li li:hover  {
  background: transparent;
  border-color: #fff;
  }
```

Once again, this seems like a lot of CSS, but we still have that problem of having to write our declarations for each individual browser. The second level of menu items is also complicated by the need to override or undo any styles we applied to the top level of the menu that we don't want to apply here. For example, we applied a `float` property to all items at the top level of our menu, but we had to undo that for the second level of the menu.

I'm sure you're starting to see why most of the support issues for the Superfish plugin are CSS related, rather than JavaScript related. There's a lot to keep track of here.

3. Finally, we still have a third level of menu to style. Just like the second level, we need to undo any cascading styles that we don't want to apply. Add the following styles to your `styles.css` file:

```
/**** Level 3 ****/
ul.sf-menu li.sfHover li ul,
ul.sf-menu li:hover li ul  {
  background: #cb2d79;
  top: -999em;
  -webkit-border-radius: 7px;
  -webkit-border-top-left-radius: 0;
  -moz-border-radius: 7px;
  -moz-border-radius-topleft: 0;
  border-radius: 7px;
  border-top-left-radius: 0;
  }

ul.sf-menu li.sfHover li ul li,
ul.sf-menu li:hover li ul li  {
  background: transparent;
  border: 0 none;
  }
```

```
ul.sf-menu li li.sfHover ul,
ul.sf-menu li li:hover ul  {
  left: 9em; /* match ul width */
  top: 0;
  }

.sf-menu li.sfHover li.sfHover li a,
.sf-menu li:hover li:hover li a  {
  background: transparent;
  }

.sf-menu li li li:first-child a  {
  padding-top: 14px;
  }

.sf-menu li li li a:hover  {
  background: transparent;
  color: #fff;
  }

/*** ARROWS ***/
.sf-sub-indicator  {
  display: none;
  }
```

And take a deep breath, because we've finally reached the end of the CSS to create a custom style for the menu. Don't worry, this was a particularly complex design using lots of new CSS3 styles. If you pick something a bit simpler, it could be a lot less code that you'll have to create to get the style working.

The bonus of this CSS is that it will work even without enabling JavaScript. The Superfish plugin just enhances the menu and makes it more usable.

What just happened?

We wrote custom CSS to style our menu to match a design that we created. In order to get hover states working correctly, we had to remember to style both the `:hover` pseudoclass and the `.sfHover` class. We also had to dig into the cascading feature of CSS and decide which styles should cascade down through all levels of the menu and which should not. And finally, we had to keep in mind that newer CSS3 properties have to be declared in different ways for different browsers—for now, at least. All of this adds up to a drop-down menu requiring more custom CSS than you might expect at first. Just be patient and keep the cascade in mind as you work down through the levels of the menu.

Custom animation

Now that we've got the CSS for our custom style written, let's take a look at customizing the animation that shows the submenus. A sliding animation would be better suited to my menu style. The default animation is to fade the submenus in, but I'd rather override this default behavior and replace it with a sliding animation.

Time for action – incorporating custom animation

Follow these steps to incorporate custom animations to your menu:

1. Fading the menu in means that the menu opacity is animating from 0 to 100 percent. I'd rather animate the height of the submenu, so that the submenu slides into view. To do that, open your scripts.js file and we'll customize the animation value inside the `superfish()` method:

   ```
   $(document).ready(function(){
     $('#sfNav').superfish({
       animation:  {height:'show'}
     });
   });
   ```

   Just adding a value here will override the default behavior of the plugin and replace it with the animation we choose instead.

2. Now when you refresh the page in a browser, you'll see the submenus slide into view instead of fade in, which is a much more fitting animation for the CSS I've used to style the menus.

What just happened?

We took advantage of one of the customization options for the Superfish plugin to change the show animation of the nested subnavigation links. There are more customization options covered in the documentation of the Superfish menu.

The hoverIntent plugin

Earlier, I pointed out that one problem with our menu was how quickly the menu reacted to the `mouseover` event. Any time the mouse is moved over the menu, the nested menus open. While that might seem like a good thing at first, it might be disconcerting or surprising to site visitors if they are simply moving their mouse on the screen and aren't intending to use the drop-down or fly-out menu.

The Superfish plugin has built-in support for the hoverIntent plugin. The hoverIntent plugin sort of pauses the `mouseover` event and makes the page wait to see if the mouse slows down or stops on an item to make sure it's what the site visitor intended to do. That way if the site visitor just happens to roll his/her mouse over the drop-down menu on their way to something else on the page, the submenus won't start appearing, throwing them into confusion.

If you'll recall, the hoverIntent plugin was actually included in the ZIP file when we downloaded the Superfish plugin.

Time for action – adding the hoverIntent plugin

Follow these steps to take advantage of the hoverIntent plugin for your menu:

1. In the Superfish download, locate the `hoverIntent.js` file inside the `js` folder and copy the file to your own `scripts` folder.
2. Next, we need to attach the hoverIntent plugin to our HTML page.

> Don't forget to keep dependencies in mind when attaching multiple JavaScript files to a page. All jQuery plugins depend on jQuery to operate, so jQuery needs to be attached to your page before any plugins. In this case, the Superfish plugin depends upon the hoverIntent plugin, so we need to make sure hoverIntent is added to our page before the Superfish plugin.

3. Add the new `<script>` tag to the bottom of your page with the other scripts as follows:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/hoverIntent.js"></script>
<script src="scripts/superfish.js"></script>
<script  src="scripts/scripts.js">
</script>
</body>
</html>
```

Now if you refresh the page in a browser, you'll see that there's a short pause when your mouse moves over the menu before the nested submenu appears. And if you run your mouse across the page quickly, crossing the menu, no unwanted submenus appear on the page.

Have a go hero – set your own speed

Try using the different customization options for the Superfish plugin that are outlined in the documentation to adjust the speed of the animation that shows the submenus.

Summary

Whew! That was a lot of work we just did, but I have to say we have a pretty impressive navigation menu to show for our efforts. We learned how to use the Superfish jQuery plugin to produce horizontal drop-down menus or vertical fly-out menus. And we learned how to fully customize the look and feel of our menu to fit our site design perfectly. Being able to hide subsections of the site until they're needed makes a complex navigation structure less overwhelming for your site visitors. It's simple and clear to see what the main sections of the site are, and they can easily drill down to just the content they want.

Next, we'll take a look at spiffing up our animation even more by super-powering it with Ajax.

7
Navigating Asynchronously

Websites are often set up so that all pages of the site share a common header and footer with only the content in between changing from page to page. Sometimes there is also one or more sidebars on the left and/or right side of the main content area that stay the same throughout the site as well. Why make our site visitors re-download the same header, footer, and sidebar content over and over again while they browse our site?

In this chapter, we'll cover the following topics:

- Setting up a website to navigate asynchronously
- Enhancing asynchronous navigation to make it more user friendly

Simple asynchronous navigation

In the early days of the Web, one solution to the repeated identical content download problem was frames. If you're too new to web development to remember, frames presented a way to break a single-page view into several different HTML files—navigating through the site involved reloading one or more of the frames while the others stayed the same. Frames helped a website to load faster and made a site easier to maintain, but in the end, they created more problems than they solved. Framed websites were easily broken, were difficult for search engines to index, often broke the back and forward buttons, and made it difficult or impossible for the site visitors to bookmark pages, share links, or print content. Because of all these problems, the use of frames has fallen out of favor.

More recently, single-page applications have started to become more popular. If you log into your Twitter account and start clicking around, you'll notice that the whole page refreshes only rarely—most of the interactions take place inside one page. If you visit any of the Gawker Media sites, you'll notice that after the initial page loads, the entire page isn't refreshed again as you browse around the site. Let's take a look at how we can accomplish this same type of interaction on our own site in a progressively enhanced way to make sure our site still works without JavaScript and can be easily indexed by search engines.

Time for action – setting up a simple website

We're going to get started by building out a small and simple website with a few pages. They'll all share the same header, navigation, sidebar, and footer. They'll all have a main content area where the unique content for each page will be displayed.

1. Get started by setting up an `index.html` file with all the associated files and folders as we did in *Chapter 1*, *Designer, Meet jQuery*. The body of the `index.html` file will contain our header, navigation, sidebar, and footer:

```
<div id="ajax-header">
  <h1>Miniature Treats</h1>
  <ul id="ajax-nav">
    <li><a href="index.html">Home</a></li>
    <li><a href="cupcakes.html">Cupcakes</a></li>
    <li><a href="petitfours.html">Petits Fours</a></li>
    <li><a href="teacakes.html">Tea Cakes</a></li>
    <li><a href="muffins.html">Muffins</a></li>
  </ul>
</div>
<div id="main-col">
  <div id="main-col-wrap">
    <p>Welcome to the miniature treats roundup. We've got a 
variety of miniature goodies to share with you.</p>
    <p>Don't be shy - just dive right in. Your mouth will water 
with the possibilites.</p>
    <p>If it's tiny enough to be a single portion all on it's own, 
we've included it here.</p>
  </div>
</div>
<div id="side-col">
  <div class="widget">
    <h4>More Information</h4>
    <ul>
```

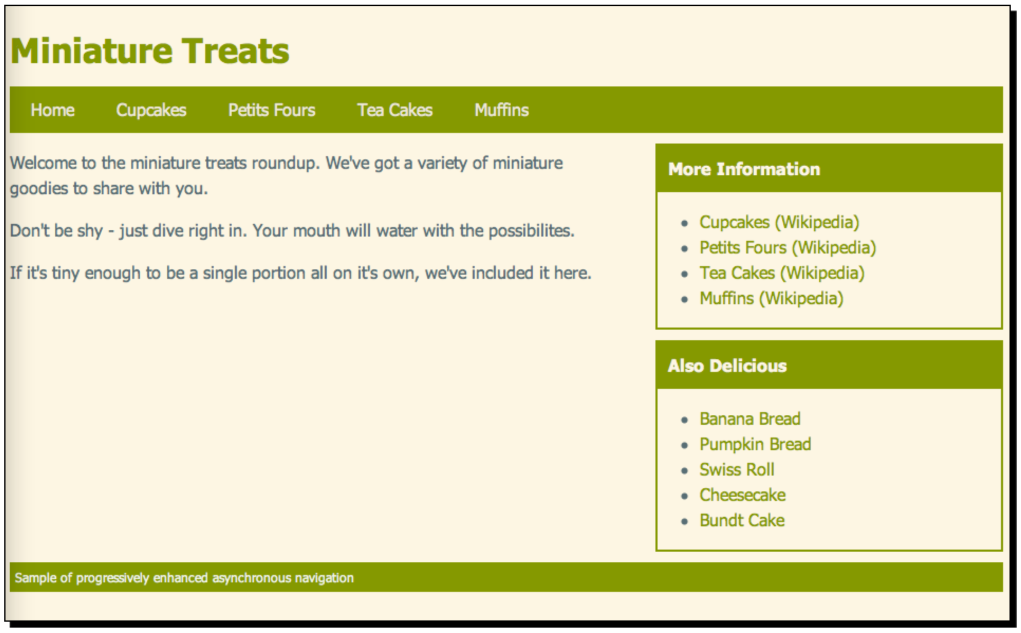
```
      <li><a href="http://en.wikipedia.org/wiki/Cupcakes">Cupcakes
(Wikipedia)</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Petit_
fours">Petits Fours (Wikipedia)</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Tea_cake">Tea
Cakes (Wikipedia)</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Muffins">Muffins
(Wikipedia)</a></li>
    </ul>
  </div>
  <div class="widget">
    <h4>Also Delicious</h4>
    <ul>
      <li><a href="http://en.wikipedia.org/wiki/Banana_
bread">Banana Bread</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Pumpkin_
bread">Pumpkin Bread</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Swiss_roll">Swiss
Roll</a></li>
      <li><a href="http://en.wikipedia.org/wiki/
Cheesecake">Cheesecake</a></li>
      <li><a href="http://en.wikipedia.org/wiki/Bundt_cake">Bundt
Cake</a></li>
    </ul>
  </div>
</div>
<div id="ajax-foot">
  <p>Sample of progressively enhanced asynchronous navigation</p>
</div>
```

You'll notice one extra `<div>` that you may not have been expecting: inside `<div>` with an `id` of `main-col`, I've added a `<div>` tag with an `id` of `main-col-wrap`. This is not used for layout or CSS purposes, but will be used once we create our JavaScript for asynchronously loading the content.

2. Next, we'll write some CSS to create a simple layout. Open your `styles.css` file and add the following styles:

```
#ajax-header    { margin: 40px 0 0 0; }
#ajax-header h1    { color:#859900;margin:0 0 10px 0;padding:0; }
#ajax-nav    { background:#859900;margin:0;padding:0;overflow:hidd
en;zoom:1; }
#ajax-nav li    { list-style-type:none;margin:0;padding:10px
20px;display:block;float:left; }
#ajax-nav a,
#ajax-nav a:link,
#ajax-nav a:visited  { color: #eee8d5; }
#ajax-nav a:hover,
#ajax-nav a:active  { color: #fff; }
#main-col    { float:left;width:60%; }
#side-col    { float:right;width:35%; }
.widget  { border:2px solid #859900;margin:10px 0; }
.widget h4  { margin:0 0 10px 0;padding:10px;background:#859900;co
lor:#FDF6E3; }
.float-right    { float:right;margin:0 0 10px 10px; }
.float-left  { float:left;margin:0 10px 10px 0; }
.source  { font-size:12px; }
#ajax-foot  { clear:both;margin:10px 0 40px 0;padding:5px;backgrou
nd:#859900;color:#f3f6e3; }
#ajax-foot p    { margin:0;padding:0;font-size:12px;}
```

The final page will look similar to the following screenshot:


Miniature Treats

Home Cupcakes Petits Fours Tea Cakes Muffins

Welcome to the miniature treats roundup. We've got a variety of miniature goodies to share with you.

Don't be shy - just dive right in. Your mouth will water with the possibilites.

If it's tiny enough to be a single portion all on it's own, we've included it here.

More Information

- Cupcakes (Wikipedia)
- Petits Fours (Wikipedia)
- Tea Cakes (Wikipedia)
- Muffins (Wikipedia)

Also Delicious

- Banana Bread
- Pumpkin Bread
- Swiss Roll
- Cheesecake
- Bundt Cake

Sample of progressively enhanced asynchronous navigation

If you're feeling inspired, feel free to write some extra CSS to style your page to be a bit fancier.

3. Next, we'll create the other pages of the site, namely the pages for cupcakes, petits fours, tea cakes, and muffins. The HTML will be exactly the same as the home page, with the exception of the content inside `<div>` with an `id` of `main-col-wrap`. The following is a sample of my content for the cupcakes page:

```
<div id="main-col-wrap">
  <h2>Cupcakes</h2>
  <p><img src="images/cupcakes.jpg" class="float-right" alt="Photo
of cupcakes"/>A cupcake is a small cake designed to serve one
person, frequently baked in a small, thin paper or aluminum cup.
As with larger cakes, frosting and other cake decorations, such as
sprinkles, are common on cupcakes.</p>
  <p>Although their origin is unknown, recipes for cupcakes have
been printed since at least the late 18th century.</p>
  <p>The first mention of the cupcake can be traced as far back
as 1796, when a recipe notation of "a cake to be baked in small
cups" was written in <em>American Cookery</em> by Amelia Simms.
The earliest documentation of the term <em>cupcake</em> was in
"Seventy-five Receipts for Pastry, Cakes, and Sweetmeats" in 1828
in Eliza Leslie's <em>Receipts</em> cookbook.</p>
  <p class="source">Text source: <a href="http://en.wikipedia.org/
wiki/Cupcakes">Wikipedia</a><br/>Image source: <a href="http://
flickr.com/people/10506540@N07">Steven Depolo</a> via <a
href="http://commons.wikimedia.org/wiki/File:Blue_cupcakes_for_
graduation,_closeup_-_Tiffany,_May_2008.jpg">Wikimedia Commons</
a></p>
</div>
```

Outside of this `<div>`, the rest of my page is exactly the same as the home page we created earlier. Go ahead and create the pages for muffins, tea cakes, and petits fours in a similar manner so that you have a five-page website with a shared header, navigation, sidebar, and footer.

Don't forget that each page of your site should contain a link to the `styles.css` file in the head section and a link to jQuery and the `scripts.js` file at the bottom of the document, just before the closing `</body>` tag.

What just happened?

We set up a simple five-page website in HTML. Each page of our website shares the same header, navigation, sidebar, and footer. Then we set up some simple CSS to style our page. The only hint that something fancy is going to happen here is an extra `<div>` wrapped around our main content area—the area of the page that contains different content from page to page.

Time for action – adding Ajax magic

If you click around this small and simple site in your browser, you'll see that we're reloading the same header, navigation, sidebar, and footer over and over again. Only the content in the main content area of the page is changing from page to page. Let's use the magic of jQuery to fix that.

1. Just a reminder that these Ajax functions won't work unless your pages are being served by a server. To see this code in action, you'll either have to upload your pages to a server or create a server on your own computer. First, we'll open our `scripts.js` file and set to work writing our code. We'll get started as we often do with the document ready statement as follows:

```
$(document).ready(function(){
  // Our code will go here
});
```

2. We'll need to select all the links inside our navigation. That will look similar to this:

```
$(document).ready(function(){
  $('#ajax-nav a')
});
```

3. When those links are clicked by the site visitor, the browser responds by loading the requested page. That's the behavior that we'd like to override, so we'll bind a function to the links that overrides the link's click behavior as follows:

```
$(document).ready(function(){
  $('#ajax-nav a').bind('click', function(){
    // Our clicky code goes here
  });
});
```

4. The first thing we need to do when a site visitor clicks a link is cancel the default behavior. We can do that by telling the function to return `false`:

```
$(document).ready(function(){
  $('#ajax-nav a').bind('click', function(){
    return false;
  });
});
```

Now if you reload your simple site in the browser, you'll see that clicking on the links in the main navigation doesn't do anything. The page you requested no longer loads into the browser. We've set the stage for our own code.

5. If we're going to fetch a page from the server, we need to know which page we're fetching. We need to know which URL we need to call. Luckily, our links already contain this information in their `href` attributes. For example, by looking at the HTML for our cupcakes link:

   ```
   <a href="cupcakes.html">Cupcakes</a>
   ```

   we can see that the page we need to request to get information on cupcakes is `cupcakes.html`.

   We're going to use jQuery to get the `href` attribute of the link that was just clicked:

   ```
   $(document).ready(function(){
     $('#ajax-nav a').bind('click', function(){
       var url = $(this).attr('href');
       return false;
     });
   });
   ```

   Now we have a variable called `url` that contains the `href` attribute of the link that was clicked. Remember, variables are just containers. If our site visitor has just clicked the cupcakes link, then the `url` variable will contain `cupcakes.html`. If the site visitor has just clicked the muffins link on the other hand, then the `url` variable contains `muffins.html`. This function gets called each time the site visitor clicks on any of the links in the main navigation - `$(this)` will always refer to the link that was just clicked.

6. Now that we know which page on the server contains the information the site visitor has requested, what do we do with it? Luckily, jQuery provides us with the `load()` method which makes loading content from the server into our page very easy. We're going to select the element on the page where we'd like to load our content and then call the `load()` method for that element. In this case, we're going to select the `<div>` tag with the `id` of `main-col`, since that's the container of the content that changes from page to page:

   ```
   $(document).ready(function(){
     $('#ajax-nav a').bind('click', function(){
       var url = $(this).attr('href');
       $('#main-col').load();
       return false;
     });
   });
   ```

7. If you reload the page in the browser and click on links in the main navigation, you'll see that nothing happens. No errors are reported by the browser—so what's the problem?

Remember Maggie the dog who was eating bacon back in *Chapter 1*, *Designer, Meet jQuery* Maggie had an eat method like this:

```
Maggie.eat();
```

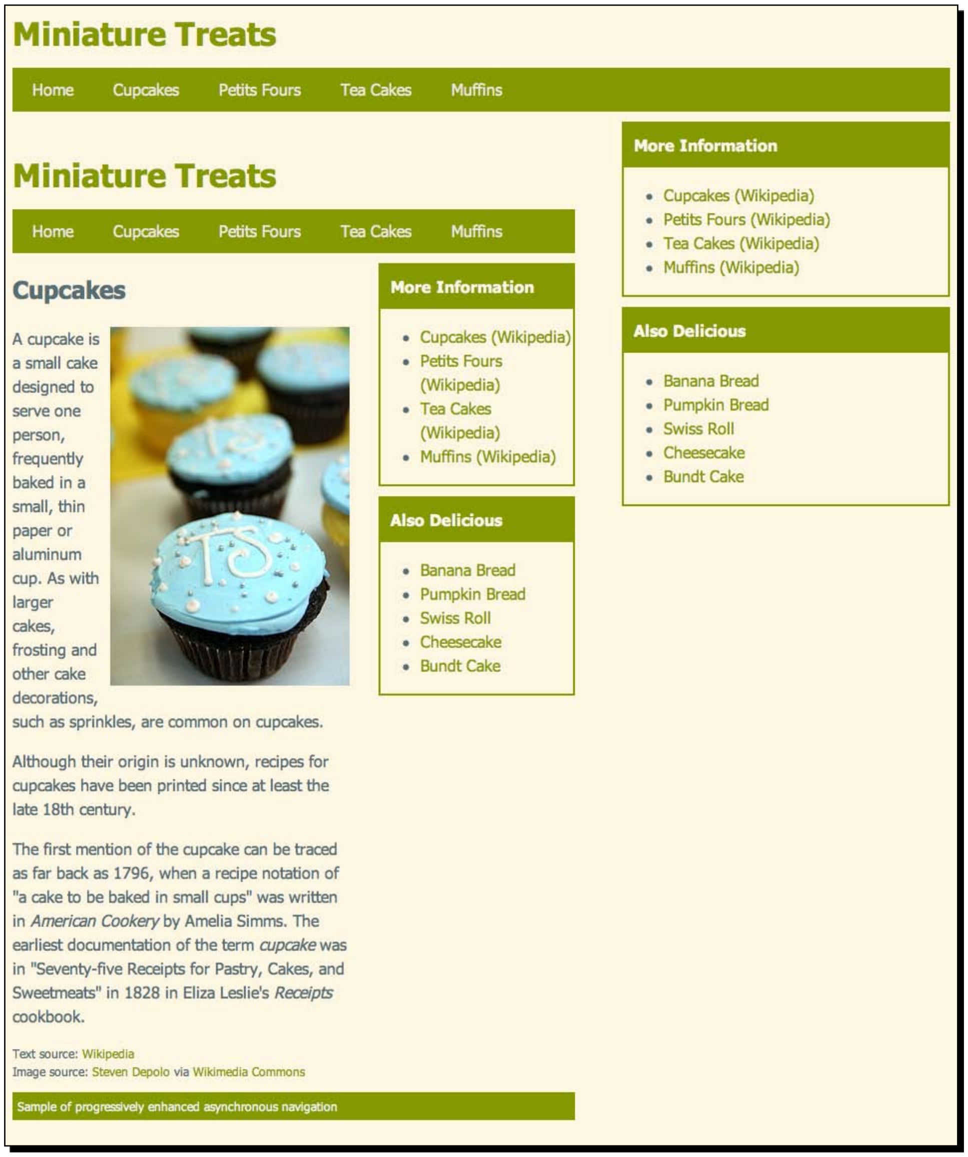
However, remember, she couldn't just eat—she had to eat something. So we passed `bacon` to the `eat()` method of Maggie as follows:

```
Maggie.eat('bacon');
```

The `load` method is similar. We can't just load—we have to load something. In this case, we know what we need to load in—the content at the URL contained in our `url` variable:

```
$(document).ready(function(){
  $('#ajax-nav a').bind('click', function(){
    var url = $(this).attr('href');
    $('#main-col').load(url);
    return false;
  });
});
```

Now if you refresh the browser and try clicking on the cupcakes link in the main navigation, you'll see that the content of the cupcakes page is indeed loaded into our `#main-col` div. However, it's not quite what we had in mind, because it's loading up the entire page:



Miniature Treats

Miniature Treats

Cupcakes

A cupcake is a small cake designed to serve one person, frequently baked in a small, thin paper or aluminum cup. As with larger cakes, frosting and other cake decorations, such as sprinkles, are common on cupcakes.

Although their origin is unknown, recipes for cupcakes have been printed since at least the late 18th century.

The first mention of the cupcake can be traced as far back as 1796, when a recipe notation of "a cake to be baked in small cups" was written in *American Cookery* by Amelia Simms. The earliest documentation of the term *cupcake* was in "Seventy-five Receipts for Pastry, Cakes, and Sweetmeats" in 1828 in Eliza Leslie's *Receipts* cookbook.

Text source: Wikipedia
Image source: Steven Depolo via Wikimedia Commons

Sample of progressively enhanced asynchronous navigation

More Information

- Cupcakes (Wikipedia)
- Petits Fours (Wikipedia)
- Tea Cakes (Wikipedia)
- Muffins (Wikipedia)

Also Delicious

- Banana Bread
- Pumpkin Bread
- Swiss Roll
- Cheesecake
- Bundt Cake

More Information

- Cupcakes (Wikipedia)
- Petits Fours (Wikipedia)
- Tea Cakes (Wikipedia)
- Muffins (Wikipedia)

Also Delicious

- Banana Bread
- Pumpkin Bread
- Swiss Roll
- Cheesecake
- Bundt Cake

8. We don't want to get the whole page. We just need the content inside the `#main-col` div. That's where that extra wrapper element, `<div>` with an `id` of `main-col-wrap` comes in. We can tell jQuery to only load that `<div>` and its content into `#main-col <div>` as follows:

```
$(document).ready(function(){
  $('#ajax-nav a').bind('click', function(){
    var url = $(this).attr('href');
    $('#main-col').load(url + ' #main-col-wrap');
    return false;
  });
});
```

   This is sometimes referred to as jQuery's **partial load method**, since we're not loading the entire contents of what we've fetched into the page, just the part we care about. If you refresh the page in the browser and click through the main navigation, you'll see that the content loads up as we expected now and only the main content area of the page refreshes. The header, navigation, sidebar, and footer remain on the page while the main content area reloads.

What just happened?

We used jQuery's powerful CSS-based selectors to select all the links in the main navigation. We identified the click behavior of the links as the behavior we needed to override to get the result that we wanted. We bound a click function to the links that would run each time a link was called. We cancelled the link's default behavior of loading up a new page in the browser window. Next, we examined the link to get the URL contained in the `href` attribute. We selected the container on the page where we wanted to load up new content, and used jQuery's `load()` method to call the content we needed. We passed a selector to the `load()` method along with the URL so that jQuery would know we wanted only the content inside that selected element to load rather than the entire page.

We turned our simple website into a single-page application. And we did it using progressive enhancement so that site visitors who don't have JavaScript enabled will be able to use our site without any problems. Search engines will also be able to index the content of our site. And we did all that with just a few lines of JavaScript—thanks to jQuery!

Deluxe asynchronous navigation

You'd be downright pleased with yourself for taking an ordinary site and turning it into a single-page application in just a few lines of code, but let's face it: our simple asynchronous navigation leaves a little bit to be desired and could definitely use a bit of polish.

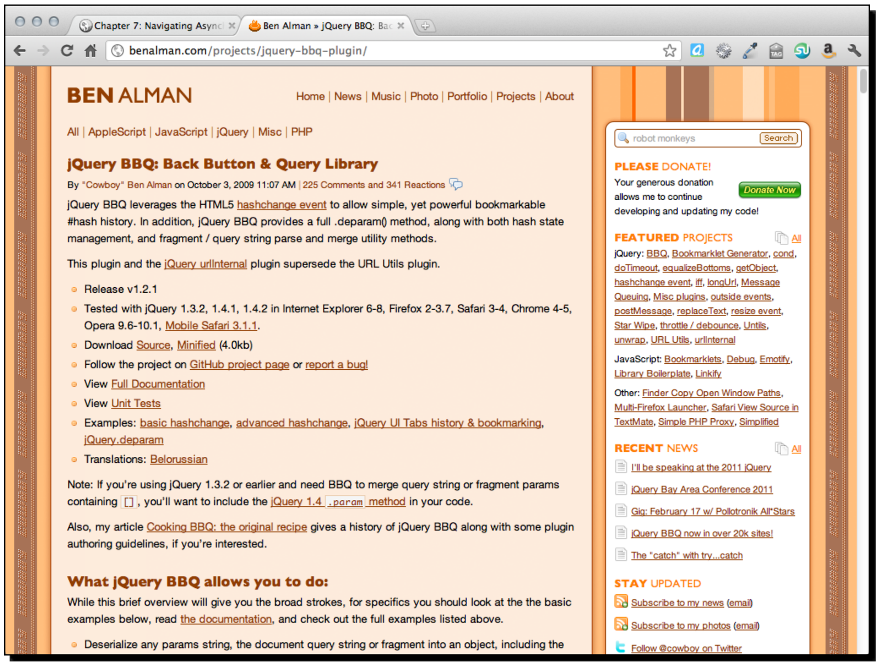
Perhaps most obviously, we've broken the back and forward buttons in the browser. We can no longer use them to navigate between the pages of our site. We've also taken away our site visitor's ability to bookmark or share a link to a page of our site. We also don't give any feedback to our site visitor that anything is happening after they've clicked a link in our main navigation. Since our pages are short and simple, they should usually load up pretty quickly, but the Internet is notoriously unpredictable in the speed department. Sometimes it could take a half second, a full second, or more to load up our content—and our site visitor has no idea their browser is hard at work trying to get the new content for them—it just looks like nothing's happening.

There are a few other nice touches we can add to make the whole thing nicer and faster too, so let's get started on the deluxe version of asynchronous navigation.

Time for action – building deluxe asynchronous navigation

To add some missing functionality to our asynchronous navigation, we're going to use Ben Alman's excellent jQuery BBQ plugin. Even though that name might make you feel hungry, BBQ stands for Back Button and Query in this case. We'll keep working with the files we created in the last example.

1. First, we'll need to get a copy of the BBQ plugin to work with. Head over to `http://benalman.com/projects/jquery-bbq-plugin/` to get the download file and the documentation and examples for the jQuery BBQ plugin.



Chapter 7: Navigating Async
Ben Alman » jQuery BBQ: Bac
benalman.com/projects/jquery-bbq-plugin/
BEN ALMAN
Home | News | Music | Photo | Portfolio | Projects | About
All | AppleScript | JavaScript | jQuery | Misc | PHP
robot monkeys
Search
Donate Now

jQuery BBQ: Back Button & Query Library

By "Cowboy" Ben Alman on October 3, 2009 11:07 AM | 225 Comments and 341 Reactions

jQuery BBQ leverages the HTML5 hashchange event to allow simple, yet powerful bookmarkable #hash history. In addition, jQuery BBQ provides a full .deparam() method, along with both hash state management, and fragment / query string parse and merge utility methods.

This plugin and the jQuery urlInternal plugin supersede the URL Utils plugin.

- Release v1.2.1
- Tested with jQuery 1.3.2, 1.4.1, 1.4.2 in Internet Explorer 6-8, Firefox 2-3.7, Safari 3-4, Chrome 4-5, Opera 9.6-10.1, Mobile Safari 3.1.1.
- Download Source, Minified (4.0kb)
- Follow the project on GitHub project page or report a bug!
- View Full Documentation
- View Unit Tests
- Examples: basic hashchange, advanced hashchange, jQuery UI Tabs history & bookmarking, jQuery.deparam
- Translations: Belorussian

Note: If you're using jQuery 1.3.2 or earlier and need BBQ to merge query string or fragment params containing `[]`, you'll want to include the jQuery 1.4 `.param` method in your code.

Also, my article Cooking BBQ: the original recipe gives a history of jQuery BBQ along with some plugin authoring guidelines, if you're interested.

What jQuery BBQ allows you to do:

While this brief overview will give you the broad strokes, for specifics you should look at the the basic examples below, read the documentation, and check out the full examples listed above.

- Deserialize any params string, the document query string or fragment into an object, including the

PLEASE DONATE!

Your generous donation allows me to continue developing and updating my code!

FEATURED PROJECTS All

jQuery: BBQ, Bookmarklet Generator, cond, doTimeout, equalizeBottoms, getObject, hashchange event, iff, longUrl, Message Queuing, Misc plugins, outside events, postMessage, replaceText, resize event, Star Wipe, throttle / debounce, Untils, unwrap, URL Utils, urlInternal

JavaScript: Bookmarklets, Debug, Emotify, Library Boilerplate, Linkify

Other: Finder Copy Open Window Paths, Multi-Firefox Launcher, Safari View Source in TextMate, Simple PHP Proxy, Simplified

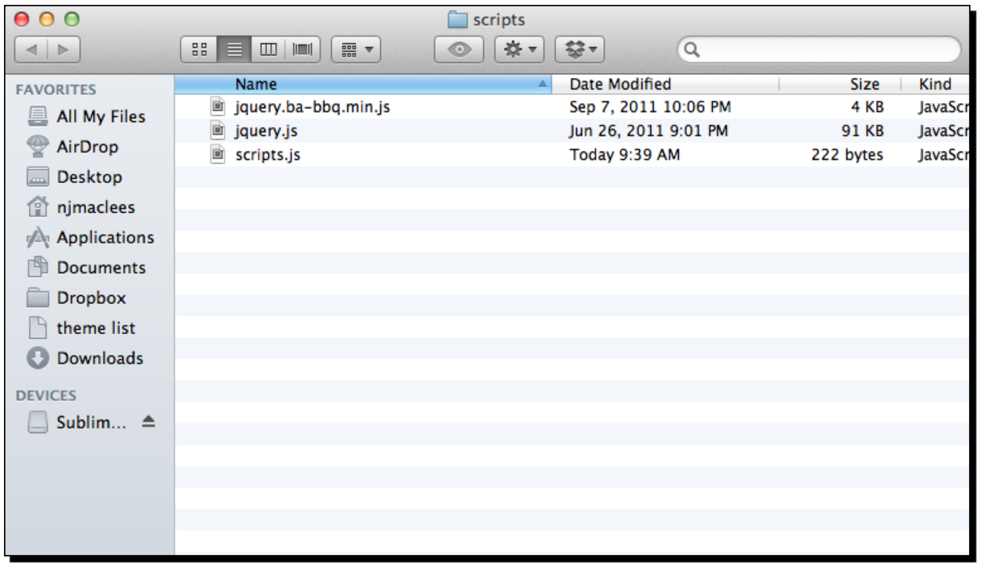
RECENT NEWS All

- I'll be speaking at the 2011 jQuery
- jQuery Bay Area Conference 2011
- Gig: February 17 w/ Pollotronik All*Stars
- jQuery BBQ now in over 20k sites!
- The "catch" with try...catch

STAY UPDATED

- Subscribe to my news (email)
- Subscribe to my photos (email)
- Follow @cowboy on Twitter

As usual, we're going to download the minified version of the plugin and drop it into our `scripts` folder alongside jQuery and our `scripts.js` file.



Name	Date Modified	Size	Kind
jquery.ba-bbq.min.js	Sep 7, 2011 10:06 PM	4 KB	JavaScr
jquery.js	Jun 26, 2011 9:01 PM	91 KB	JavaScr
scripts.js	Today 9:39 AM	222 bytes	JavaScr

2. Next, open up each of the HTML pages of your mini website and add the BBQ plugin, after jQuery and before `scripts.js`:

```
<script type="text/javascript" src="scripts/jquery.js"></script>
<script type="text/javascript" src="scripts/jquery.ba-bbq.min.
js"></script>
<script type="text/javascript" src="scripts/scripts.js"></script>
</body>
</html>
```

Now we're ready to get to work on building the deluxe version of our asynchronous navigation.

What just happened?

We downloaded the jQuery BBQ plugin and attached it to each of our pages. So far, this hasn't made a difference on our site—we've attached the BBQ plugin, but we aren't using it to do anything. Next up, we'll take a look at how to put the BBQ plugin to use.

Time for action – using the BBQ plugin

Our first order of business is to get those back and forward buttons working, and allow our site visitors to bookmark and share links to individual pages. That's why we've included the jQuery BBQ plugin.

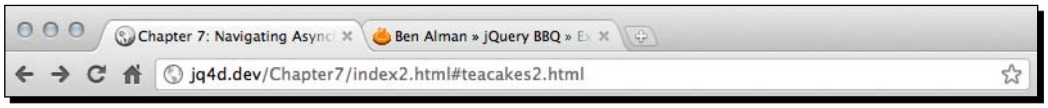
1. We're going to write some new JavaScript, so erase the code we wrote earlier in `scripts.js` and replace it with a simple document ready statement as follows:

```
$(document).ready(function(){
  // Our deluxe ajaxy code goes here
});
```

2. Next, we're going to select each of the links in our main navigation and replace the URLs with hash links so that the browser thinks they are internal to our HTML page.

```
$(document).ready(function(){
  $('#ajax-nav a').each(function(){
    $(this).attr('href', '#' + $(this).attr('href'));
  });
});
```

We're selecting all the links in the main navigation, then looping through all of them to add a `#` character at the front of the URL. For example, the `cupcakes.html` link is now `#cupcakes.html`. If you refresh the page in the browser, you'll see that clicking the links doesn't change anything on the page, but it does update the hash in the URL in the browser's location bar.


Chapter 7: Navigating Async
Ben Alman » jQuery BBQ » Ex
jq4d.dev/Chapter7/index2.html#teacakes2.html

3. Next, we're going to bind a function to the window's `hashchange` event. Modern browsers have provided an event called `hashchange` that fires whenever the URL's hash changes, just as it's doing when you click the main navigation links. Older browsers don't support the `hashchange` event, but that's where the jQuery BBQ plugin comes in. It provides support for a pseudo `hashchange` event in most browsers so that we only have to write our code once without worrying about browser differences. Here's how we bind a function to the `hashchange` event:

```
$(document).ready(function(){
  $('#ajax-nav a').each(function(){
    $(this).attr('href', '#' + $(this).attr('href'));
  });
  $(window).bind('hashchange', function(e) {
    // our function goes here
  });
});
```

4. The function we write will now be called each time the window's hash changes, which we know is going to happen each time the site visitor clicks on a link in our main navigation. Now we can write the code to tell the browser what to do when this happens.

```
$(document).ready(function(){
  $('#ajax-nav a').each(function(){
    $(this).attr('href', '#' + $(this).attr('href'));
  });
  $(window).bind('hashchange', function(e) {
    var url = e.fragment;
    $('#main-col').load(url + ' #main-col-wrap');
  });
});
```

First, we're setting up a variable called `url` and setting it equal to `e.fragment`. The `fragment` property is made available by the jQuery BBQ plugin. It's equal to the hash of the URL without the hash symbol. So if the window's hash changes to `#cupcakes.html`, `e.fragment` will be equal to `cupcakes.html`.

The next line of code is the same as our basic Ajax navigation example. I'm going to select the container on the page where I want to load my content, then call the `load()` method. I'm going to pass the URL and jQuery selector for the part of the page at that URL that I want to load into the browser.

If you refresh the page in the browser now, you'll see that our main navigation is again working asynchronously. Clicking a link loads up only the main content area of the page while the rest remains unchanged. There is one important difference, though—if you click the back and forward buttons, they work. Once you've clicked through to the cupcakes page, you can click the back button to return to the home page.

5. There's just one thing left to do to get our navigation optimized and that's to make sure that our site visitors can bookmark and share links to our pages. If you click on the cupcakes page, copy the URL from the browser's location bar, and open either a new browser window or a new tab and paste in the URL, you'll see that you get the site's home page rather than the cupcake page. If you look at the URL, the `#cupcakes.html` hash is there, we just have to tell our code to look for it. The simplest way to do that is to fire the window's `hashchange` event as soon as the page loads in the browser. Here's how we do that:

```
$(document).ready(function(){
  $('#ajax-nav a').each(function(){
    $(this).attr('href', '#' + $(this).attr('href'));
  });
  $(window).bind('hashchange', function(e) {
    var url = e.fragment;
```

```
          $('#main-col').load(url + ' #main-col-wrap');
        });
        $(window).trigger('hashchange');
      });
```

Now, you can open up that cupcakes link in a new window and you'll see the cupcakes page load up, just as it should. Our `hashchange` function fires as soon as the page is loaded, which loads in the correct content.

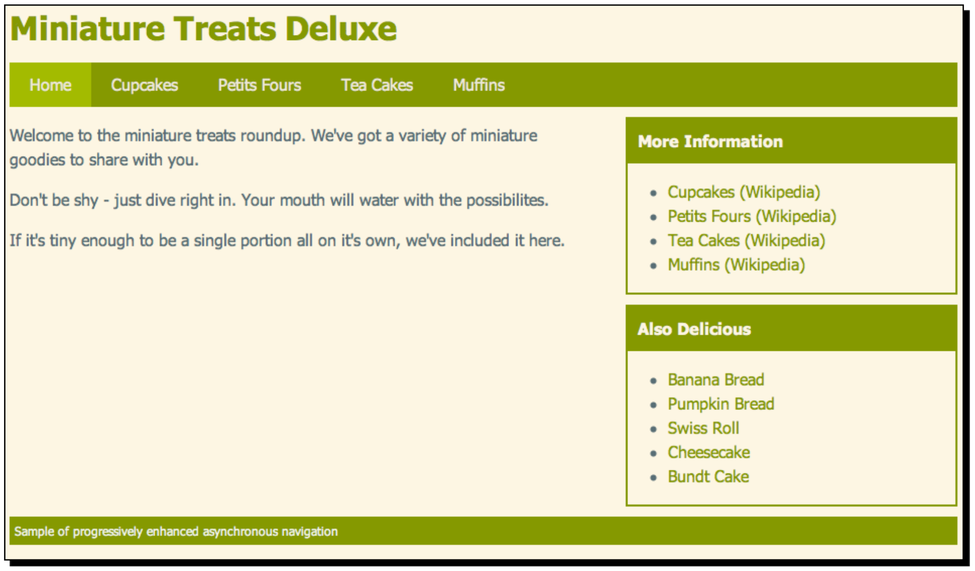
What just happened?

We used jQuery to loop through each of our navigation links and replace them with internal links or hash links. Why not just do this in HTML? Because we want to make sure that our page continues to work for users with JavaScript disabled.

Then we used the jQuery BBQ plugin to change our asynchronous navigation to enable both bookmarking and sharing of links and the back and forward buttons in the browser. This allows our site to behave just like a single-page application without breaking the site visitor's expected experience.

Time for action – highlighting the current page in the Navigation

We've already made our asynchronous navigation much better than our simple example, but I think we can keep going and make it even better. Next up, we're going to highlight the page currently being viewed in the navigation to make it easy for our site visitors to see which page they're on.



Welcome to the miniature treats roundup. We've got a variety of miniature goodies to share with you.

Don't be shy - just dive right in. Your mouth will water with the possibilites.

If it's tiny enough to be a single portion all on it's own, we've included it here.

1. First up, let's open up `styles.css` again and write a `.current` CSS class for the navigation:

   ```
   #ajax-nav li.current{ background:#a3bb00; }
   ```

   I've made my navigation bar green, so I'm going to make the `.current` class a slightly lighter shade of green so that the current item is highlighted in the menu. You can follow my example or create your own style—whatever suits your taste.

2. Now we just need to apply our `.current` class to the current navigation item. We're going to add a few lines to the `hashchange` event function we wrote earlier. We'll start by checking to see if there's a hash in the window location:

   ```
   $(document).ready(function(){
     $('#ajax-nav a').each(function(){
       $(this).attr('href', '#' + $(this).attr('href'));
     });
     $(window).bind('hashchange', function(e) {
       var url = e.fragment;
       $('#main-col').load(url + ' #main-col-wrap');
       if (url) {
         // The code if there is a hash
       } else {
         // The code if there is not a hash
       }
     });
     $(window).trigger('hashchange');
   });
   ```

3. Now, if there is a hash, then we want to find the link in my main navigation that corresponds to the hash, find its parent container, and add the current class. That sounds like a lot, but I can do that in one line:

   ```
     $(window).bind('hashchange', function(e) {
       var url = e.fragment;
       $('#main-col').load(url + ' #main-col-wrap');
       $('#ajax-nav li.current').removeClass('current');
       if (url) {
         $('#ajax-nav a[href="#' + url + '"]').parents('li').
   addClass('current');
       } else {
         // The code if there is not a hash
       }
     });
   ```

I'm using jQuery's powerful attribute selectors to select the link with the `href` attribute equal to the window's hash. Then I'm using the `parents()` method to get the link's parents. I'm passing `li` to the `parents()` method to tell jQuery I'm only interested in one parent, the `<li>` that contains my link. Then I'm using the `addClass()` method to add my current class to the current link.

4. If there isn't a hash, then I want to highlight the home page, which is the first page in our main navigation. I'll select the first `<li>` and add the current class as shown in the following code:

```
  $(window).bind('hashchange', function(e) {
    var url = e.fragment;
    $('#main-col').load(url + ' #main-col-wrap');
    $('#ajax-nav li.current').removeClass('current');
    if (url) {
      $('#ajax-nav a[href="#' + url + '"]').parents('li').
addClass('current');
    } else {
      $('#ajax-nav li:first-child').addClass('current');
    }
  });
```

5. Now, if you refresh the page in the browser and click through the pages, you'll see that the current page is highlighted, but as you move through the site, more and more of the navigation is highlighted—we're not removing the old highlight before adding a new one. We'll add this line to remove the current highlight before adding a new one:

```
  $(window).bind('hashchange', function(e) {
    var url = e.fragment;
    $('#main-col').load(url + ' #main-col-wrap');
    $('#ajax-nav li.current').removeClass('current');
    if (url) {
      $('#ajax-nav a[href="#' + url + '"]').parents('li').
addClass('current');
    } else {
      $('#ajax-nav li:first-child').addClass('current');
    }
  });
```

Refresh the page in the browser and you'll see that the highlight is now working as it should, highlighting only the current page.

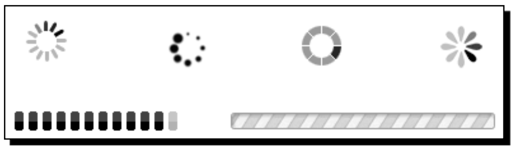
What just happened?

We added a few lines of code to our `hashchange` function to add a highlight to the current page in the navigation. This will help the site visitor orient themselves on the site and further enforce their current location.

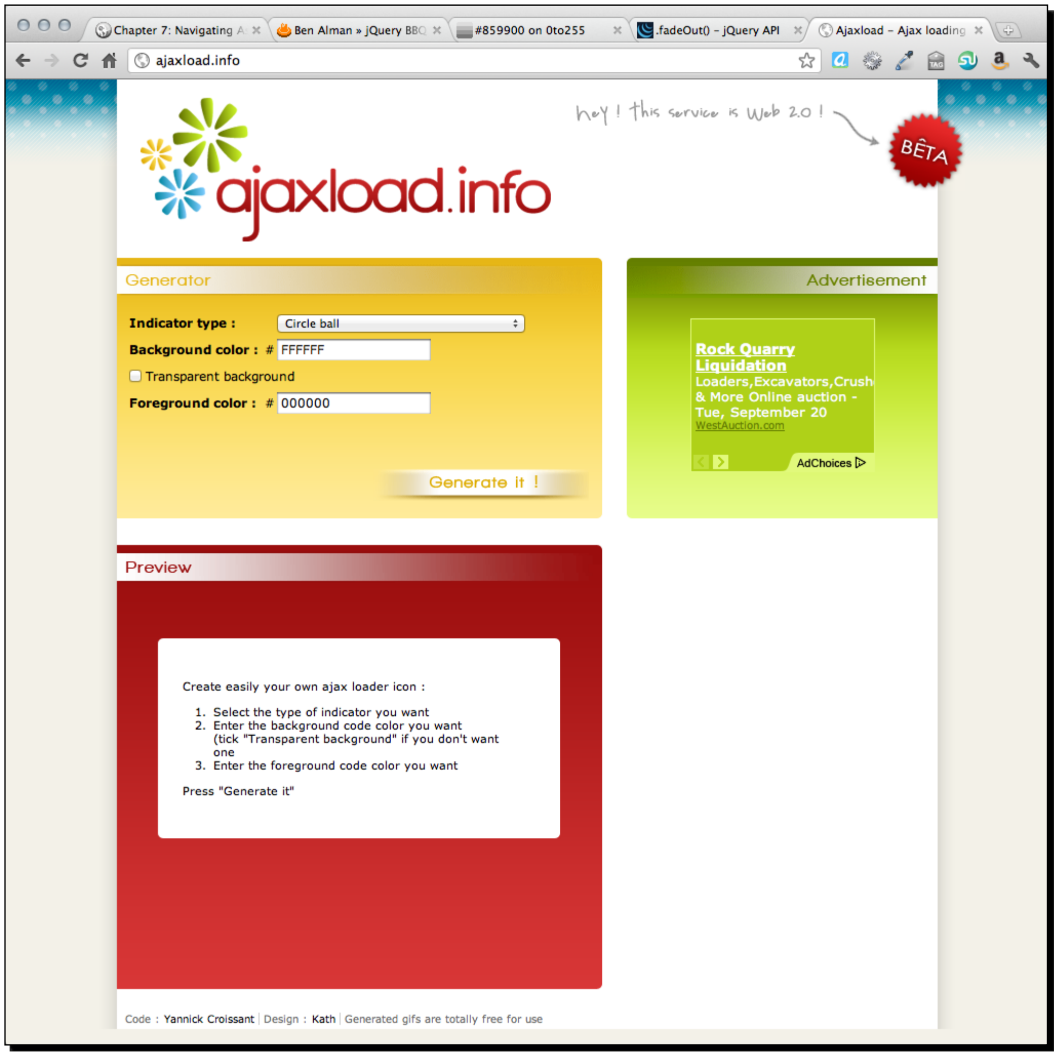
Time for action – adding a loading animation

Next, we'd like to show the site visitor that something is happening as soon as they click on the link in the navigation. Remember that if the response from the server is slow, the site visitor can't see that anything is happening. Even though the browser is hard at work fetching the content for the new page, there's no indication to the site visitor that anything is happening. Let's add in a little animation to make it obvious that something's happening on our page.

Loading animations can take many different forms: spinning daisies, animated progress bars, blinking dots—anything that will communicate that there's an action in progress will help make your site feel snappier and more responsive for your site visitors.



1. First, head over to `http://ajaxload.info` to create and download a loading animation of your choice.



Indicator type : Circle ball
Background color : # FFFFFF
Transparent background
Foreground color : # 000000

Rock Quarry Liquidation
Loaders,Excavators,Crush & More Online auction - Tue, September 20
WestAuction.com

Create easily your own ajax loader icon :

1. Select the type of indicator you want
2. Enter the background code color you want (tick "Transparent background" if you don't want one
3. Enter the foreground code color you want

Press "Generate it"

Code : Yannick Croissant | Design : Kath | Generated gifs are totally free for use

2. Select the type, background color, and foreground color in the **Generator** box, then click on the **Generate It!** button.

3. You'll see a preview of your button in the **Preview** box along with a link to download your button. Click on the **Download It!** link to download a copy of the loading animation you just created.

4. After you've downloaded your button, drop it in your `images` folder along with the other images you're using on your website.

5. Now, let's think through the modifications we have to make to our page. We want to fade out the content that is currently shown in the `#main-col` div and show our loading animation in its place until the server sends back the content for our new page. As soon as we get that content back, we want to hide the loading animation and display the content.

   When we're ready to show the loading animation to our site visitors, we want it to be visible immediately. It would be no good at all if we were to have to go and fetch the image from the server—the actual page content might be returned before our image. So we'll have to preload the image. Using jQuery, that's very simple. As soon as the document has loaded into the browser, we'll create a new image element as shown in the following code:

```
$(document).ready(function(){
  var loadingImage = $('<img src="/images/ajax-loader.gif"/>');
  $('#ajax-nav a').each(function(){
...
```

   Just creating this element is enough to preload the image into the browser's cache. Now when we're ready to show the image, it will be available immediately without waiting.

6. Next, we have to write a bit of CSS to handle how our loading image is displayed. We'll wrap it in a simple paragraph tag to which we'll add a bit of padding and center the image:

```
#loading      { padding:20px;text-align:center;display:none; }
```

7. Note that we're also setting the `display` to `none`—that way we won't have the image showing up until we're ready for it. We only want our animation to appear if the URL has a hash, so inside our `if`/`else` statement, we'll append the loading animation to the `#main-col` div:

```
...
    if (url) {
      $('#main-col').append('<p id="loading"></p>')
      $('#loading').append(loadingImage);
      $('#ajax-nav a[href="#' + url + '"]').parents('li').
addClass('current');
    } else {
...
```

We've added a paragraph with the `id` of `loading` to the document and we've appended our pre-loaded loading image to that paragraph. Remember, even though it's there, it's not visible yet, since we've hidden it with CSS.

8. Next, we'll fade out the content that's currently showing on the page. In case our content returns quickly from the server, we want to make sure we're not getting in the way, so we'll tell the animation to complete quickly:

```
...
    if (url) {
      $('#main-col').append('<p id="loading"></p>')
      $('#loading').append(loadingImage);
      $('#ajax-nav a[href="#' + url + '"]').parents('li').
addClass('current');
      $('#main-col-wrap').fadeOut('fast');
    } else {
...
```

9. Finally, we want to show our loading animation, but we don't want it to appear until after the content has faded out. To make sure it doesn't show up before then, we'll add it as a callback function to the `fadeOut()` method. A callback function is a function that's called after the animation completes. Here's how we add a callback function to the `fadeOut()` method:

```
...
    if (url) {
      $('#main-col').append('<p id="loading"></p>')
      $('#loading').append(loadingImage);
      $('#ajax-nav a[href="#' + url + '"]').parents('li').
addClass('current');
      $('#main-col-wrap').fadeOut('fast', function(){
        $('#loading').show();
      });
    } else {
...
```

Now, when the site visitor clicks a link, the hash in the location bar will update. That will fire off our code to fade the page's current content out, show a loading animation, and then replace the loading animation with the new page content as soon as it's returned by the server. If you're really lucky, your site visitor won't even get a chance to see the loading animation because your server will return the new page content quickly. However, if there's a slowdown anywhere along the way, your site visitor will get a clear message that something's happening and they won't be left wondering or feeling like your site is slow and unresponsive.

What just happened?

We added some animation effects to show the site visitor that something was happening in the event that the server's response with the new page content was delayed more than a fraction of a second. The site visitor will immediately see the content fade out and a loading animation take its place until the server responds with the new page content.

If you're looking at your pages from your local computer using WAMP or MAMP, chances are the new content will be returned so quickly you won't get a chance to see the loading animation. However, if you upload your pages to a server and access them via the Internet, you're almost guaranteed to see the loading animation for at least a fraction of a second while the browser fetches the new content.

Summary

In this chapter, we learned how to set up a simple website and then we enhanced it to behave like a single-page application without breaking it for search engines or site visitors who have JavaScript disabled. First, we set up a simple version that might be suitable for use in some simple cases. Then we took a look at setting up the deluxe version that allowed for bookmarking and sharing of links, working back and forward buttons, current page highlighting in the navigation, and smooth transition animations to show the site visitor the browser was hard at work. All of this was relatively simple and straightforward thanks to jQuery and the jQuery BBQ plugin.

Next up, we'll take a look at loading content into lightboxes.

8
Showing Content in Lightboxes

It has become common to see galleries of photos displayed in lightboxes on the web. Lightboxes can be useful for other things too—playing videos, showing additional information, displaying important information to site visitors, or even showing other websites. In this chapter, we'll cover how to use the flexible and adaptable Colorbox plugin to create lightboxes for a variety of purposes.

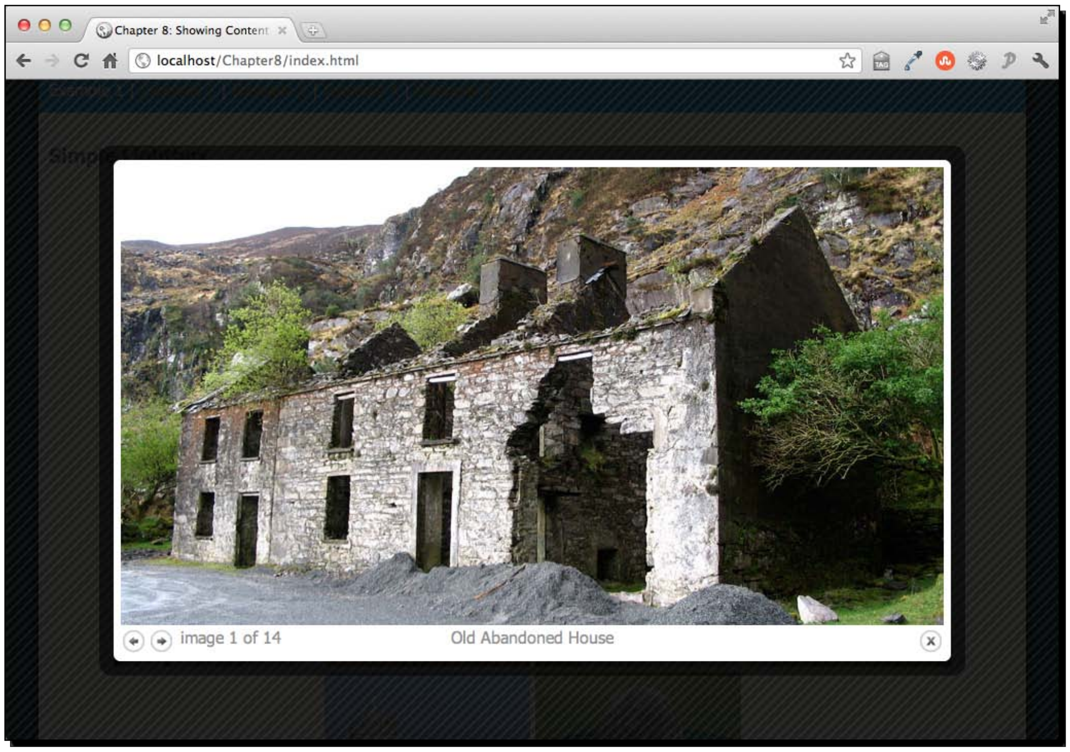
In this chapter, we'll take a look at how to use the Colorbox plugin to:

- Create a simple photo gallery
- Customize photo gallery settings
- Build a fancy login box
- Play a collection of videos
- Create a one-page website portfolio

Simple photo gallery

A simple photo gallery is probably the most common use for lightboxes. We'll set up a page that shows thumbnails of each photo and displays the full-size image in a lightbox when the thumbnail is clicked. To get started, you'll need a series of photographs with smaller size thumbnails of each.

Here's an example of a photo displayed in a lightbox:



Time for action – setting up a simple photo gallery

We'll walk through creating a simple photo gallery with the Colorbox plugin:

1. We'll get started by setting up a basic HTML page and associated files and folders just like we did in *Chapter 1*, *Designer, Meet jQuery*. The body of the HTML document will contain a list of thumbnails:

```
<ul class="thumb-list">
       <li><a href="images/abandoned-house.jpg" title="Old
Abandoned House" rel="ireland"><img src="images/thumbs/
abandoned-house.jpg" alt="Abandoned House"/></a></li>
```

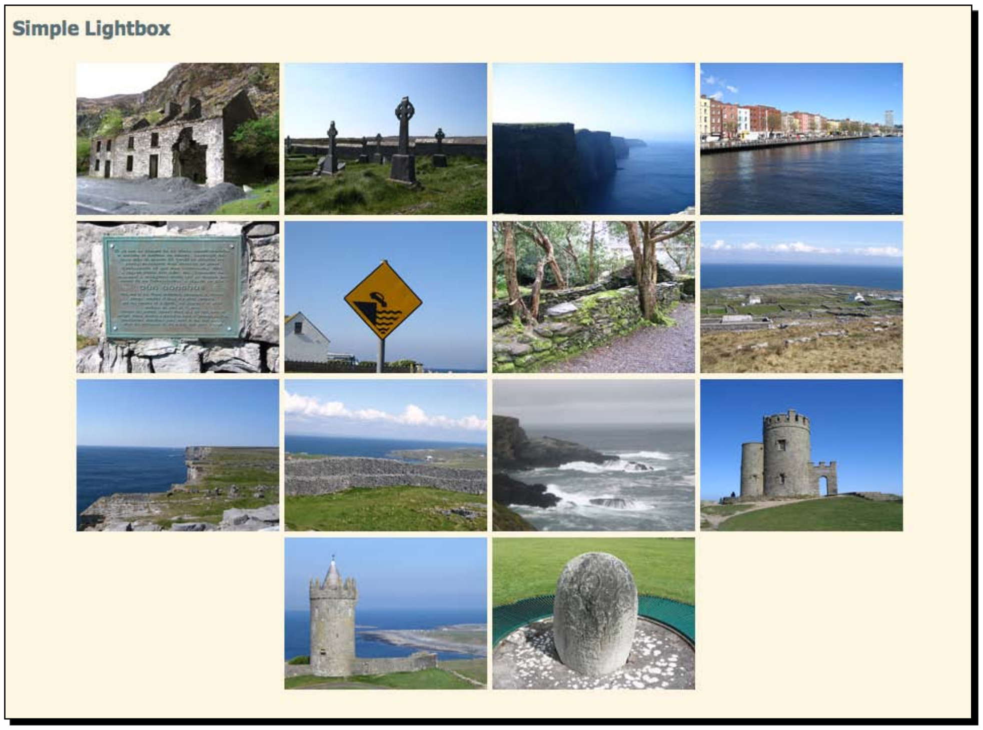
```
        <li><a href="images/cemetary.jpg" title="Celtic Cemetary
with Celtic Crosses" rel="ireland"><img src="images/thumbs/
cemetary.jpg" alt="Celtic Cemetary"/></a></li>
        <li><a href="images/cliffs-of-moher.jpg" title="Cliffs of
Moher" rel="ireland"><img src="images/thumbs/cliffs-of-moher.jpg"
alt="Cliffs of Moher"/></a></li>
        <li><a href="images/dublin.jpg" title="River Liffey
in Dublin" rel="ireland"><img src="images/thumbs/dublin.jpg"
alt="Dublin"/></a></li>
        <li><a href="images/dun-aonghasa.jpg" title="Dun Aonghasa on
Inis More" rel="ireland"><img src="images/thumbs/dun-aonghasa.jpg"
alt="Dun Aonghasa"/></a></li>
        <li><a href="images/falling-in.jpg" title="Warning Sign"
rel="ireland"><img src="images/thumbs/falling-in.jpg" alt="Falling
In"/></a></li>
        <li><a href="images/guagan-barra.jpg" title="Guagan
Barra" rel="ireland"><img src="images/thumbs/guagan-barra.jpg"
alt="Guagan Barra"/></a></li>
        <li><a href="images/inis-more.jpg" title="Stone Fences on
Inis More" rel="ireland"><img src="images/thumbs/inis-more.jpg"
alt="Inis More"/></a></li>
        <li><a href="images/inis-more2.jpg" title="Cliffs on Inis
More's West Coast" rel="ireland"><img src="images/thumbs/inis-
more2.jpg" alt="Inis More Too"/></a></li>
        <li><a href="images/inis-more3.jpg" title="Inis More Fence"
rel="ireland"><img src="images/thumbs/inis-more3.jpg" alt="Inis
More Three"/></a></li>
        <li><a href="images/mizen-head.jpg" title="Crashing Waves
Near Mizen Head" rel="ireland"><img src="images/thumbs/mizen-head.
jpg" alt="Mizen Head"/></a></li>
        <li><a href="images/obriens-tower.jpg" title="O'Brien's
Tower at the Cliffs of Moher" rel="ireland"><img src="images/
thumbs/obriens-tower.jpg" alt="O'Brien's Tower"/></a></li>
        <li><a href="images/random-castle.jpg" title="Some Random
Castle" rel="ireland"><img src="images/thumbs/random-castle.jpg"
alt="Random Castle"/></a></li>
        <li><a href="images/turoe-stone.jpg" title="Turoe Stone"
rel="ireland"><img src="images/thumbs/turoe-stone.jpg" alt="Turoe
Stone"/></a></li>
</ul>
```

Note that we've wrapped each thumbnail in a link to the full-size version of the image. If you load the page in a browser, you'll see that the page works for users with JavaScript disabled. Clicking a thumbnail opens the full-size image in the browser. The back button takes you back to the gallery.

Note that we've also included a `title` attribute on each link. This is helpful for our site visitors as it will show a short description of the image in a tooltip when they hover over the thumbnail with their mouse, but it will also be used later on for the Colorbox plugin. We've also included a `rel` attribute on each link and set it equal to `ireland`. This will make selecting our group of links to Ireland images easy when we're ready to add the Colorbox plugin magic.

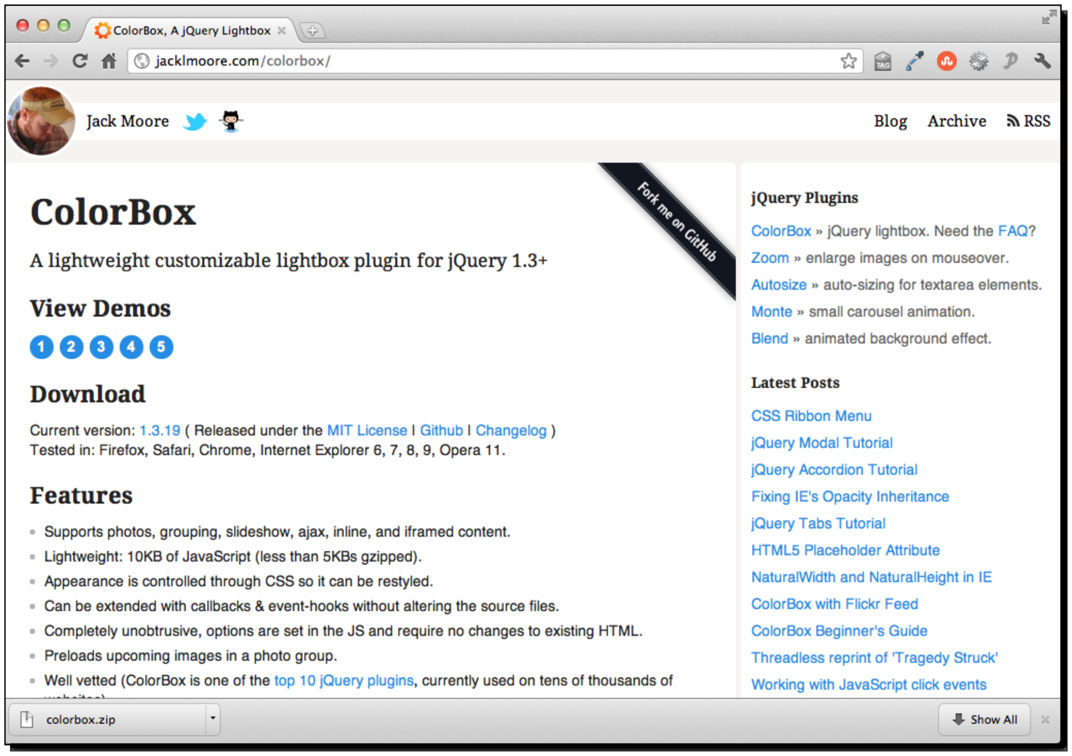
2. Next, we'll add a bit of CSS to lay our images out in a grid. Open your `styles.css` file and add these styles:

```
ul.thumb-list       { margin:20px 0;padding:0;text-align:center; }
ul.thumb-list li    { margin:0;padding:0;display:inline-block; }
```



Feel free to play around a bit with the CSS to create a different layout for your image thumbnails if you'd like.

3. Now, let's add the jQuery magic. We're going to be using Color Powered's Colorbox plugin. Head over to `http://jacklmoore.com/colorbox` to find the downloads, documentation, and demos. You'll find the download link in the **Download** section, near the top of the page. Just click the current version number to download a ZIP file.



ColorBox

A lightweight customizable lightbox plugin for jQuery 1.3+

View Demos

Download

Current version: 1.3.19 (Released under the MIT License | Github | Changelog)
Tested in: Firefox, Safari, Chrome, Internet Explorer 6, 7, 8, 9, Opera 11.

Features

- Supports photos, grouping, slideshow, ajax, inline, and iframed content.
- Lightweight: 10KB of JavaScript (less than 5KBs gzipped).
- Appearance is controlled through CSS so it can be restyled.
- Can be extended with callbacks & event-hooks without altering the source files.
- Completely unobtrusive, options are set in the JS and require no changes to existing HTML.
- Preloads upcoming images in a photo group.
- Well vetted (ColorBox is one of the top 10 jQuery plugins, currently used on tens of thousands of

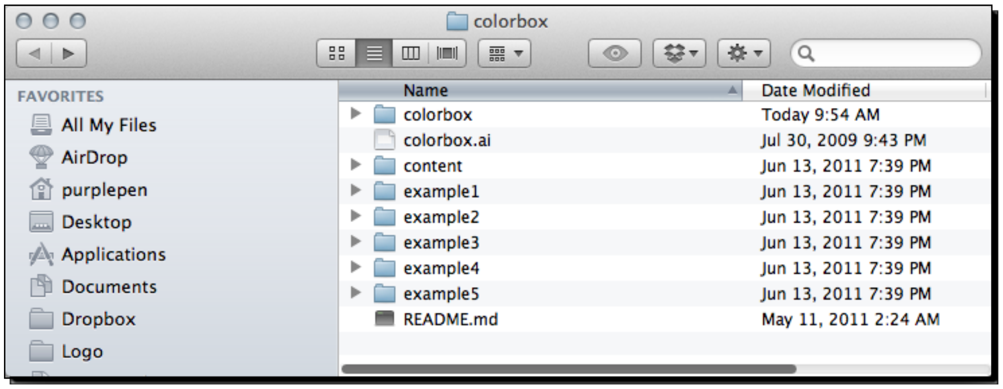
jQuery Plugins

ColorBox » jQuery lightbox. Need the FAQ?
Zoom » enlarge images on mouseover.
Autosize » auto-sizing for textarea elements.
Monte » small carousel animation.
Blend » animated background effect.

Latest Posts

CSS Ribbon Menu
jQuery Modal Tutorial
jQuery Accordion Tutorial
Fixing IE's Opacity Inheritance
jQuery Tabs Tutorial
HTML5 Placeholder Attribute
NaturalWidth and NaturalHeight in IE
ColorBox with Flickr Feed
ColorBox Beginner's Guide
Threadless reprint of 'Tragedy Struck'
Working with JavaScript click events

4. Unzip the folder and have a look inside. You'll find the plugin script file itself, of course, but a lot of other goodies as well.



Name	Date Modified
colorbox	Today 9:54 AM
colorbox.ai	Jul 30, 2009 9:43 PM
content	Jun 13, 2011 7:39 PM
example1	Jun 13, 2011 7:39 PM
example2	Jun 13, 2011 7:39 PM
example3	Jun 13, 2011 7:39 PM
example4	Jun 13, 2011 7:39 PM
example5	Jun 13, 2011 7:39 PM
README.md	May 11, 2011 2:24 AM

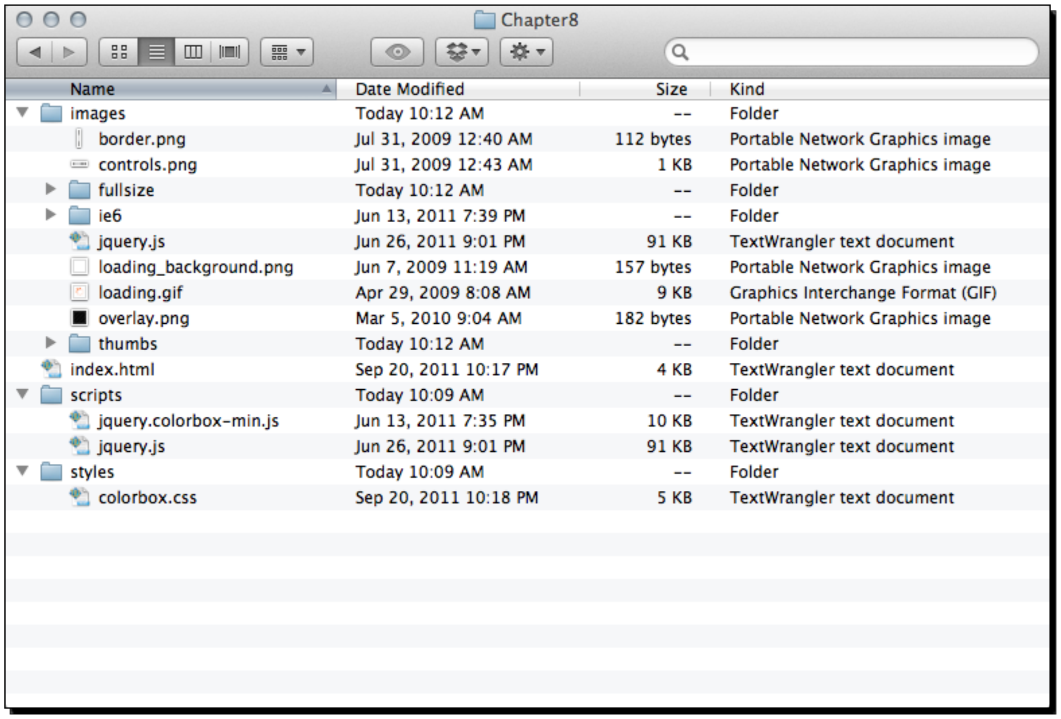
The plugin code itself is in the `colorbox` folder—you'll find both the development and minified versions. The five example folders each contain an example file (`index.html`) that shows the plugin in action. Why five different folders? Each folder contains the same basic example, but with five different looks for the Colorbox. These same examples can be viewed on the Colorbox website by clicking numbers in the **View Demos** section on the website.

Right out of the box, the plugin's developers are providing us with five different possibilities for our Colorbox's look and feel. And if that's not enough choice, they've also included a `colorbox.ai` (**Adobe Illustrator**) file that contains all of the image assets used to create these five different looks. You can customize them to your heart's content and then export your new fully custom look from Illustrator to create your own appearance. Changing colors and special effects is straightforward enough, but remember that if you change the size and shape of the image assets, you'll have to touch up the accompanying CSS file to accommodate the new sizes.

5. Try out each of the different examples, either on the website or using the example files included in the ZIP download, and note that the appearance, size, placement of the back and forward buttons, the close button, the caption, the pagination indicator (image 1 of 3), and so on, are all controlled via CSS—not the plugin code itself. This makes it very easy to customize the look and feel—it's all done via CSS rather than in JavaScript.

6. Inside the ZIP download, in the `colorbox` folder, you'll find the plugin code – a file named `jquery.colorbox-min.js`. Copy this file to your own `scripts` folder.

7. We'll get started by choosing one of the provided CSS skins. Pick your favorite, then copy and paste its CSS file to your own `styles` folder. Open up the `images` folder for that CSS skin and copy and paste the images from that folder to your own `images` folder. Once you've chosen a skin, your own setup should look like the following:



Name	Date Modified	Size	Kind
images	Today 10:12 AM	--	Folder
border.png	Jul 31, 2009 12:40 AM	112 bytes	Portable Network Graphics image
controls.png	Jul 31, 2009 12:43 AM	1 KB	Portable Network Graphics image
fullsize	Today 10:12 AM	--	Folder
ie6	Jun 13, 2011 7:39 PM	--	Folder
jquery.js	Jun 26, 2011 9:01 PM	91 KB	TextWrangler text document
loading_background.png	Jun 7, 2009 11:19 AM	157 bytes	Portable Network Graphics image
loading.gif	Apr 29, 2009 8:08 AM	9 KB	Graphics Interchange Format (GIF)
overlay.png	Mar 5, 2010 9:04 AM	182 bytes	Portable Network Graphics image
thumbs	Today 10:12 AM	--	Folder
index.html	Sep 20, 2011 10:17 PM	4 KB	TextWrangler text document
scripts	Today 10:09 AM	--	Folder
jquery.colorbox-min.js	Jun 13, 2011 7:35 PM	10 KB	TextWrangler text document
jquery.js	Jun 26, 2011 9:01 PM	91 KB	TextWrangler text document
styles	Today 10:09 AM	--	Folder
colorbox.css	Sep 20, 2011 10:18 PM	5 KB	TextWrangler text document

The `index.html` file contains the HTML with thumbnail images that link to full-size versions. The `images` folder contains the images provided with my chosen Colorbox skin, alongside my own images for my slideshow, both the thumbnail and full-size versions. My `scripts` folder contains jQuery (`jquery.js`) and the Colorbox plugin script (`jquery.colorbox-min.js`). My `styles` folder contains the CSS file for the Colorbox skin I chose.

8. We do have to open up `colorbox.css` to make a minor set of edits. In the example files, the CSS file is not in a `styles` or `css` folder, but rather sits at the top level alongside the `index.html` file. We've chosen to follow our preferred convention and store our CSS in our `styles` folder. This means that we'll have to open up the `colorbox.css` file and update the references to images in the CSS. I'll have to replace references like this:

```
#cboxTopLeft{width:21px; height:21px; background:url(images/
controls.png) no-repeat -100px 0;}
```

with references like this:

```
#cboxTopLeft{width:21px; height:21px; background:url(../images/
controls.png) no-repeat -100px 0;}
```

I'm just telling the CSS to go up one level and then look for the `images` folder. You should be able to replace all of these quickly by using the Find and Replace functionality of your text editor.

9. Next, open up your `index.html` file and attach the `colorbox.css` file in the head section, before your own `styles.css`:

```
<head>
       <title>Chapter 8: Showing Content in Lightboxes</title>
       <link rel="stylesheet" href="styles/colorbox.css"/>
       <link rel="stylesheet" href="styles/styles.css"/>
</head>
```

10. And then head down to the bottom of the file, just before the closing `</body>` tag, and attach the Colorbox plugin, after jQuery and before your own `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.colorbox-min.js"></script>
<script src="scripts/scripts.js"></script>
```

11. Now, remember that `rel="ireland"` attribute we included on each of our links? We're going to use that in our JavaScript to select all of our Ireland image links for the Colorbox plugin. Open your `scripts.js` file and write the attribute selector to select all links with a `rel` attribute equal to `ireland` inside of the document ready statement:

```
$(document).ready(function(){
       $('a[rel="ireland"]')
});
```

12. The only thing left to do is call the `colorbox()` method on those links—the Colorbox plugin will take care of everything else for us:

```
<script type="text/javascript">
       $('a[rel="ireland"]').colorbox();
</script>
```

Now if you open the page in the browser and click one of the thumbnail images, you'll see the full-size image open up in a Colorbox. You can navigate through all of the full-size images without having to close the lightbox, thanks to the back and forward buttons included. You can also move between the images by pressing the left and right arrow keys on your keyboard. The pagination indicator helps you to see where you are in the collection of photos. You'll also notice that the `title` attribute included on each link gets re-used as an image caption for each image. The Colorbox can be closed by clicking the close button, clicking outside the Colorbox, or by hitting the *Esc* key on your keyboard. All in all, it's a pretty nice experience right out of the box.

What just happened?

We used the Colorbox jQuery plugin to turn a list of links to images into a lightbox that allows site visitors to navigate through the full-size images without leaving the page. We used the `title` attribute of the links to provide captions for the images. We used one of the five Colorbox styles provided with the plugin to create a nicely-designed lightbox.

Customizing Colorbox's behavior

If you take a look through the **Settings** section of the Colorbox website, you'll see that you have plenty of options for customizing how the Colorbox behaves. Let's take a look at how we can put some of these options to use. For this section, we'll keep working with the files we set up in the previous section.

Transition

First up, we'll try out the different transition effects that are available. The default transition is `elastic`. If your full-size images are all different sizes, you'll see that Colorbox uses a nice resizing animation to transition between them. The other options for transitions are `fade` and `none`. Let's take a look at how we can modify the transition.

Time for action – using a custom transition

Follow these steps to change the default transition between images:

1. For this example, we'll take a look at how to use the `fade` transition. Open your `scripts.js` file. All we have to do is pass the `fade` value for the transition key to the `colorbox()` method as follows:

   ```
   $(document).ready(function(){
         $('a[rel="ireland"]').colorbox({transition:'fade'});
   });
   ```

   Note that we've added some curly braces inside the parentheses. Inside these curly braces, we can pass in key/value pairs to customize different aspects of the Colorbox. In this case, the key is `transition` and the value is `'fade'`.

   If you reload the page in the browser, click one of the thumbnails then click the next and previous buttons to flip through the images, you'll see that the Colorbox fades out and then back in between each image.

2. What if we decided we'd rather get rid of the transitions altogether? We'd simply have to change the value for the `transition` key to `'none'`:

   ```
   $(document).ready(function(){
          $('a[rel="ireland"]').colorbox({transition:'none'});
   });
   ```

   Now if you refresh the page in the browser, you'll see that the images change without any transition effect between them.

What just happened?

We saw how to take advantage of one of the available settings with the Colorbox plugin and modified the transition between images as our site visitor moves through them.

Fixed size

In a case where the photos you're loading into the Colorbox are of widely varying different sizes, you might decide that all the resizing is distracting to the site visitors and that you want to set a fixed size for the Colorbox. That's easy to do as well, by passing in a couple more key/value pairs. Looking through the documentation, you'll see that there are lots of settings for controlling the width and height of the Colorbox. To keep things simple, we're going to use `width` and `height`.

Time for action – setting a fixed size

Follow these steps to set a fixed width and height for the Colorbox:

1. Open up your `scripts.js` file. We're going to make a few changes to our code to set a fixed `width` and `height` for the Colorbox:

   ```
   $('a[rel="ireland"]').colorbox({
          transition: 'none',
          width: '90%',
          height: '60%'
   });
   ```

   Now if you refresh the page in the browser, you'll see that the Colorbox remains the same size. No matter what size the images or the browser window is, the Colorbox will always fill 90% of the width and 60% of the height of the browser window. The images inside resize proportionally to fit into the available space if they are too large.

What just happened?

We set the `width` and `height` settings to percentage values. This is a really helpful option if you have large photos that could potentially be larger than your site visitor's browser window. Setting the `width` and `height` to percentage values ensures that in this case, the Colorbox will be 90% of the width and 60% of the height of the site visitor's browser window, no matter what size the browser window happens to be. That way if the browser window is small, the site visitor will be able to see the complete photo.

Colorbox also provides some other settings for width and height:

innerWidth/innerHeight

These keys provide `width` and `height` values for the content inside the Colorbox instead of for the Colorbox itself. This can be helpful in cases where you know the exact width and height of the actual content, for example a video player.

InitialWidth/initialHeight

Colorbox is very flexible and can be used for a variety of different content (as we'll see shortly). Setting an `intialWidth` and `initialHeight` allow you to control the size of the Colorbox before any content is loaded in. If you load in content via Ajax, it can take a few moments to load into the Colorbox. Setting `initialWidth` and `initialHeight` allows you to specify how large the Colorbox should be while you wait for the content to load in.

maxWidth/maxHeight

These keys allow you to set a maximum width and maximum height for the Colorbox. If the content is smaller, then the box will appear smaller on the screen. But when you're loading in larger contents, they won't exceed the `maxWidth` and `maxHeight` values you specify. For example, if you wanted to set up a Colorbox for images in a variety of sizes, you could allow the Colorbox to resize with fade or elastic transitions between images, but set a `maxWidth` and `maxHeight` to be sure that larger images wouldn't exceed the site visitor's browser window.

Creating a slideshow

Colorbox also provides us with an option to automatically cycle through all the images so the site visitor doesn't have to continually click the next button to see them all.

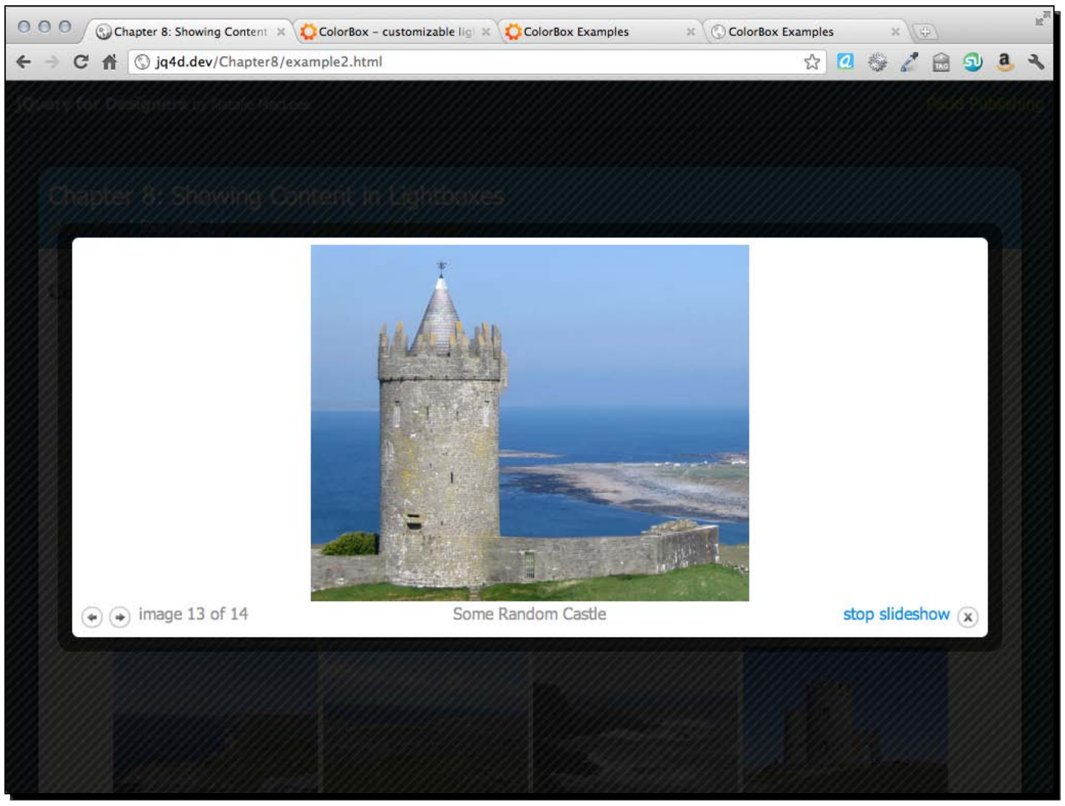
Time for action – creating a slideshow

Here's how we can turn our lightbox image gallery into a slideshow:

1. Open `scripts.js`. We're going to add another key/value pair to our settings. To create a slideshow inside our Colorbox, set the `slideshow` key to `true`:

```
$('a[rel="ireland"]').colorbox({
      transition: 'none',
      width: '90%',
      height: '60%',
      slideshow: true
});
```

   Now if you refresh the page in the browser, you'll see that after you open the Colorbox, it automatically cycles through the images, using whichever transition effect you've chosen. A link is provided so that site visitors can stop the slideshow at any time:


Chapter 8: Showing Content
ColorBox – customizable lig
ColorBox Examples
ColorBox Examples
jq4d.dev/Chapter8/example2.html
image 13 of 14
Some Random Castle
stop slideshow

2. Colorbox provides a few more keys we can use to control the slideshow. We can provide a value for `slideshowSpeed` to set the number of milliseconds each photo will be displayed. If we don't want the slideshow to automatically play, we can set `slideshowAuto` to `false`. We can change the text that appears in the link to start and stop the slideshow by passing in values for the `slideshowStart` and `slideshowStop` keys, respectively. That would all look like this:

```
$('a[rel="ireland"]').colorbox({
       transition: 'none',
       width: '90%',
       height: '60%',
       slideshow: true,
       slideshowSpeed: 2000,
       slideshowAuto: false,
       slideshowStart: 'Let\'s get started!',
       slideshowStop: 'Ok, that\'s enough.'
});
```

With this code, we've set up our slideshow to show each photo for 2 seconds (2000 milliseconds), to not start the slideshow automatically, and to customize the text on the links that start and stop the slideshow.

Note that each key/value pair is separated by a comma, but that there's no comma after the last key/value pair. No comma after the last one is only important for Internet Explorer—if you accidentally put a comma after the last key/value pair in Internet Explorer, it will throw an error and none of your JavaScripts will work. Other browsers will ignore the error and continue to work gracefully. Always test your work in Internet Explorer before you make it available to the public.

Let's talk for a minute about the \' that appears in the text I want to use for the link to start and stop the slideshow. Since these are strings, I have to wrap them in quote marks, either 'single' quote or "double" quotes will work, and it's a matter of personal preference which you choose. If I want to then use quote marks in my string, I have to escape them—which is the JavaScript way of saying I have to tell JavaScript that those are part of my string and not characters that JavaScript should pay attention to.

If I were to write my string this way:

```
slideshowStart: 'Let's get started!'
```

this would cause an error. As far as JavaScript is concerned, the ' in Let's is the closing single quote for the string – and JavaScript has no idea what to do with the rest of the line.

In this case, if my personal preference were for using double quotes for writing strings, I wouldn't have to do anything at all. This would be perfectly acceptable:

```
slideshowStart: "Let's get started!"
```

Since we're using double quotes around our string, there's no chance that JavaScript will accidentally read it as the end of our string. Once JavaScript sees an opening " character, it automatically looks for the matching ending " character.

Now that we've got our slideshow customized, refresh the page in the browser and click one of the image thumbnails to open the Colorbox. The only visible difference is the addition of the **Let's get started** link. Clicking it kicks off the slideshow and switches the link to say Ok, that's enough, so that we can stop the slideshow.

What just happened?

We saw how to create and customize a slideshow. We did this by taking the simple lightbox photo gallery and customizing it by passing a series of key/value pairs to the `colorbox()` method.

Fancy login

It's nice enough to be able to use a lightbox for displaying images and slideshows, but Colorbox is more capable and flexible than that. In this section, we'll take a look at showing a login form in a Colorbox. Note that our login form isn't hooked up to anything and won't actually function in the sample case. But this same technique can be applied to a dynamic site to allow your site visitors to view the login form in a lightbox.

Time for action – creating a fancy login form

Follow these steps to create a login form in a lightbox:

1. We'll get started by setting up an HTML page and associated files and folders like we did in *Chapter 1*, *Designer, Meet jQuery*. Our HTML page will contain a header that displays a login form. It's common for sites to enable people to login from any page on the site:

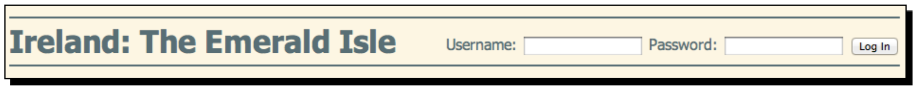
```
<div id="example-header">
       <h1>Ireland: The Emerald Isle</h1>
       <form action="#" id="login-form">
              <div><label for="username">Username:</label> <input
type="text" id="username"/></div>
              <div><label for="password">Password:</label> <input
type="text" id="password"/></div>
              <div><input type="submit" value="Log In"/></div>
```

```
        </form>
</div>
```

2. Next, we'll open `styles.css` and add some CSS, so that the header displays with the title on the left and the form on the right:

```
#example-header      { border-bottom:2px solid #586E75;
border-top:2px solid #586E75;overflow:hidden;zoom:1; }
#example-header h1   { float:left;padding:0;margin:0; }
#example-header #login-form      { float:right;padding-top:15px; }
#example-header #login-form div  { display:inline; }
#login-link                      { display:block;float:right;paddi
ng-top:15px; }
#login-link:focus    { outline:none; }
```

If you view the page in a browser, you'll see the following:



This is perfectly acceptable for users without JavaScript enabled—they'll be able to log into the site from any page. But I do think it's a bit cluttery. So if the site visitor has JavaScript enabled, we will hide the login form, and show it in a Colorbox when the site visitor is ready to log in.

3. Next, we'll get ready to use the Colorbox plugin the same way we did in the previous section: choose one of the provided styles for the Colorbox and attach its stylesheet to the head section of our document, move all the required images to your `image` directory and update the path to the images in the CSS, and attach the Colorbox plugin at the foot of the document, between jQuery and our `scripts.js` tag.

4. Once all that's out of the way, we're ready to write our JavaScript. Open up `scripts.js` and write your document ready statement:

```
$(document).ready(function(){
        //Our code goes here
});
```

5. The first thing we'd like to do is hide the login form. We're going to do that using JavaScript rather than CSS because we do want the login form to be visible for the site visitors who don't have JavaScript enabled. We want to hide the form immediately as soon as the page is loaded, so we'll write our hiding code inside the `ready()` method for the document:

```
$(document).ready(function(){
        var form = $('#login-form');
        form.hide()
});
```

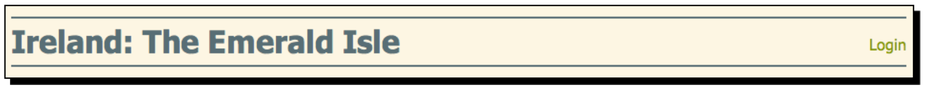
You'll notice that we created a variable called `form` and used it to store the jQuery selector for the form. We're going to have to refer to the login form several times in our code. We could write `$('#login-form')` each time we wanted to select the login form, but each time, jQuery would have to look through the DOM of the page to find it anew. If we store it in a variable, our code will run faster and be more efficient since jQuery will not have to find the login form each time we refer to it.

If you refresh the page in the browser, you'll see that the login form has disappeared.

6. But now we need a way for site visitors to be able to show it again to be able to log in. We'll use jQuery to add a login link to the page, which will appear right where the form was:

```
$(document).ready(function(){
        var form = $('#login-form');
        form.hide()
        form.before('<a href="#login-form" id="login-link">Login</
a>');
});
```

Already, we're referring to the form again—inserting the login link before the form. We already included some styles in the CSS to style the link and display it where we'd like. If you refresh the page in the browser, you'll see the login form has been replaced by a login link:



7. But clicking the login link doesn't do anything. Let's fix that by adding in some Colorbox magic. We'll select our login link and call the `colorbox()` method as follows:

```
$(document).ready(function(){
        var form = $('#login-form');
        form.hide()
        $('#login-form').before('<a href="#login-form" id="login-
link">Login</a>');
        $('#login-link').colorbox();
});
```

Refresh the page in the browser and try clicking the link. Hmmm...not really what we had in mind, right? We have to tell Colorbox that we want to load up some content that's already on the page.

8. We already put the reference to the login form in the `href` attribute of the link, so we'll use that to our advantage. We'll pass a couple of key/value pairs to the `colorbox()` method to tell Colorbox that we want to load some content that's already on the page and exactly which content we want to show:

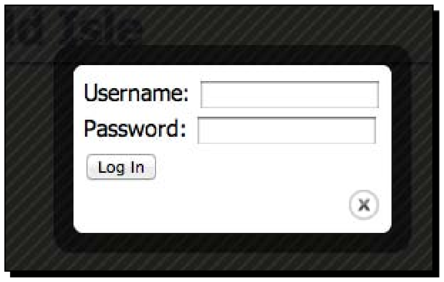
```
$(document).ready(function(){
       var form = $('#login-form');
       form.hide()
       $('#login-form').before('<a href="#login-form" id="login-
link">Login</a>');
       $('#login-link').colorbox({
              inline: true,
              content: $(this).attr('href')
       });
});
```

Refresh the page in the browser and you'll see that the Colorbox opens, but it appears to be empty. That's because we hid our form. It's been loaded into the Colorbox, but it's hidden from view.

9. We'll use another key/value pair to tell Colorbox to show the form when the Colorbox opens:

```
$(document).ready(function(){
       var form = $('#login-form');
       form.hide()
       $('#login-form').before('<a href="#login-form" id="login-
link">Login</a>');
       $('#login-link').colorbox({
              inline: true,
              content: $(this).attr('href'),
              onOpen: function(){form.show();}
       });
});
```

`onOpen` is one of the keys provided by the Colorbox plugin. It allows us to write a function that will be run when the Colorbox opens. In this case, I'm going to find my form and show it. Now if you refresh the page in the browser, you'll be able to see the form in the ColorBox as follows:



10. This looks fine enough, and we'll touch this up with a bit of CSS in a moment to make it look even better. But what happens when you close the Colorbox? That pesky login form is visible again in the header. So we'll pass another key/value pair to our `colorbox()` method to hide the form when the Colorbox closes:

```
$(document).ready(function(){
       var form = $('#login-form');
       form.hide()
       $('#login-form').before('<a href="#login-form" id="login-
link">Login</a>');
       $('#login-link').colorbox({
              inline: true,
              content: $(this).attr('href'),
              onOpen: function(){form.show();},
              onCleanup: function(){form.hide();},
       });
});
```

That will make sure our form is hidden when the Colorbox is closed so it doesn't show up in the header again.

11. Now, let's make our login form look a bit friendlier. Open up `styles.css` and add some CSS that styles the login form inside the lightbox:

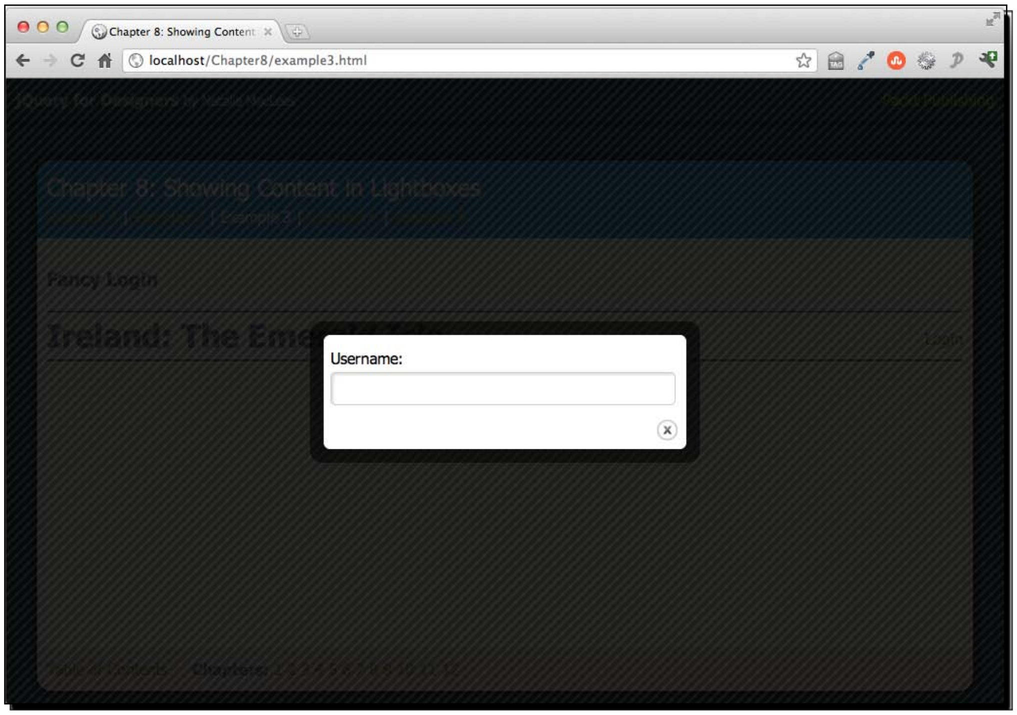
```
#cboxContent form div   { padding:5px 0; }
#cboxContent label      { display:block; }
#cboxContent input[type='text']    { font-size:1.2em;padding:5px
;width:342px;border:1px solid #ccc;box-shadow:inset 2px 2px 2px
#ddd;border-radius:5px; }
```

```
#cboxContent input[type='submit']   { font-
size:1.2em;padding:10px; }
```

12. And we also want to make the login form box a bit wider, so we're going to pass a `width` key to the `colorbox()` method as follows:

```
$(document).ready(function(){
       var form = $('#login-form');
       form.hide()
       form.before('<a href="#login-form" id="login-link">Login</
a>');
       $('#login-link').colorbox({
              width: '400px',
              inline: true,
              content: $(this).attr('href'),
              onOpen: function(){form.show();},
              onCleanup: function(){form.hide();},
       });
});
```

Now, if you refresh the page in the browser, you'll see our Colorbox is indeed 400 pixels wide and our login form has taken on the nice chunky appearance we wanted with our CSS, but there's still a bit of a problem. Our form is too tall for the Colorbox:



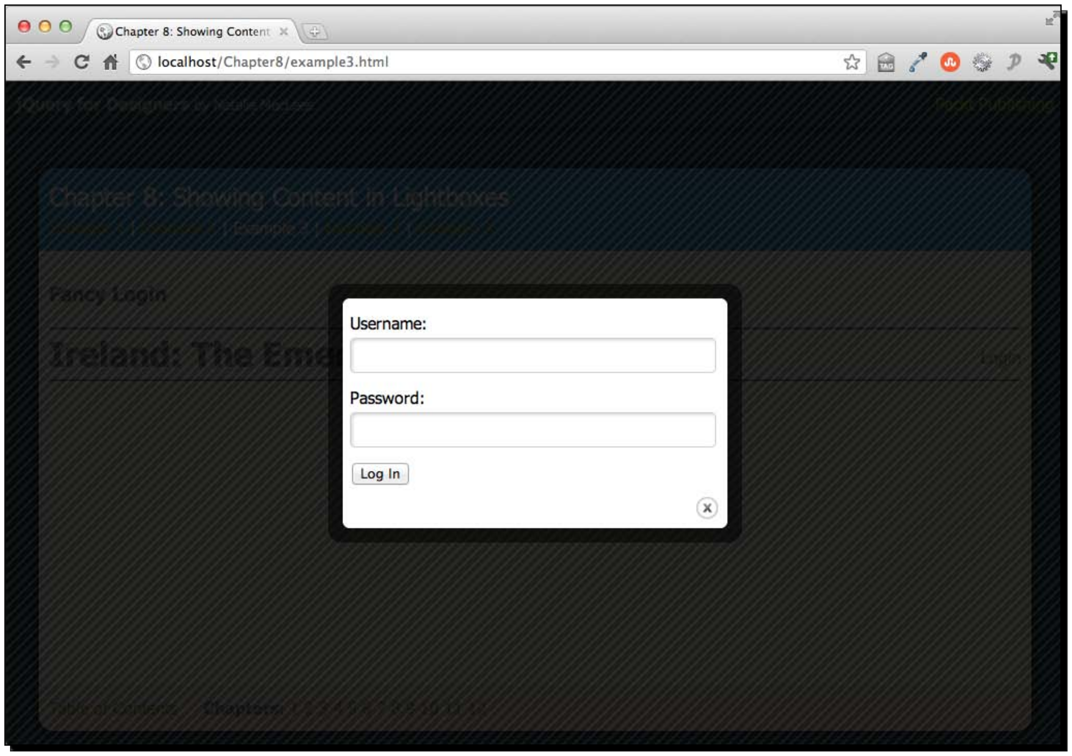
The Colorbox script hasn't realized that our form has a different set of CSS once it's displayed inside the Colorbox—it's still expecting the form to be of the same height it was when it was displayed in the header. But that form is much smaller. If you put your mouse over the login form and scroll down, you'll see the rest of the login form is there—we just can't see it.

13. We don't want any scrolling in our Colorbox, so we'll turn that off and we'll tell the Colorbox to resize itself to its content instead by passing a couple more key/value pairs to the `colorbox()` method as follows:

```
$(document).ready(function(){
       var form = $('#login-form');
       form.hide()
       form.before('<a href="#login-form" id="login-link">Login</
a>');
       $('#login-link').colorbox({
              width: '400px',
              inline: true,
              scrolling: false,
              content: $(this).attr('href'),
              onOpen: function(){form.show();},
              onComplete: function(){$.colorbox.resize();},
              onCleanup: function(){form.hide();},
       });
});
```

The scrolling key allows us to turn off any scrolling inside the Colorbox, and the `onComplete` key is a callback function that's called as soon as content loads into the Colorbox. As soon as the content loads into the Colorbox, we're going to call a method that the Colorbox plugin has made available to us to resize the Colorbox to accommodate its content.

Now, if you refresh the page in the browser, you'll see the Colorbox slide open to a larger height to accommodate the new CSS for our form. Perfect!



What just happened?

We learned how to take a simple header login form and change it to a login link that opens a login form in a Colorbox when clicked. We worked through any potential problems caused by this approach by passing in callback functions as values for keys specified in the Colorbox plugin documentation. We learned how to call functions to run when the Colorbox opens, when the content is loaded into the Colorbox, and when the Colorbox closes. We learned that we can force the Colorbox to resize to accommodate its current contents by calling the `$.colorbox.resize()` method.

Video player

Colorbox is flexible enough to be used to display a video player as content. We'll link out to a YouTube video, then add some Colorbox magic to display the video in a Colorbox. Please note that this example makes use of Ajax, and will therefore only work if you upload your files to a server or if you create a server on your own computer.

Time for action – showing a video in a lightbox

Follow these steps to set up Colorbox to play a set of videos:

1. We'll get started as we usually do, by setting up a basic HTML file and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*. In the body of our HTML document, we're going to include a link to a YouTube video:

    ```
    <p><a href="http://www.youtube.com/embed/2_HXUhShhmY?autoplay=1"
    id="video-link">Watch the video</a></p>
    ```

    Note a couple of things about my video link. First, I'm using the embed URL for the video rather than the link to YouTube's video page. For users without JavaScript enabled, this will take them to a stand-alone video player page on YouTube's site. For users with JavaScript enabled, it will ensure that only the video player is loaded into the Colorbox rather than the full YouTube video page. Second, I'm adding a parameter to the URL for the video, setting `autoplay` to `1`. This is how you can make embedded YouTube videos automatically play when the site visitor views your page. It's generally a bad idea to have a video autoplay, but in this case, the user will have already clicked a link that says **Watch the video**, so it seems like a safe bet that they'll be expecting a video to play once they've clicked that link.

2. Next, just as with the other Colorbox examples so far, you'll need to attach your chosen Colorbox skin CSS file in the head of your document, make sure the images are available, update the path to the images in the CSS if necessary, and finally attach the Colorbox plugin in the foot of the document.

3. Now, we'll open up our `scripts.js` file and get set to write up our custom JavaScript. We'll get started with the document ready statement:

    ```
    $(document).ready(function(){

    });
    ```

4. Next, we'll select the video link and call the `colorbox()` method:

    ```
    $(document).ready(function(){
           $('#video-link').colorbox();
    });
    ```

    But if we refresh the page in a browser and attempt to view the video, we get an error. That's because we're attempting to load in the video via Ajax, and because of browser security restrictions, we can't make asynchronous requests to a different server. In this case, we're trying to make a call to `http://youtube.com`, but that's not where our Colorbox page is hosted, so the browser blocks our request.

5. Luckily, we can create an `iframe` and load our external content into the `iframe`. And also luckily, Colorbox provides a way for us to do so easily. We'll just pass a key/value pair to the `colorbox()` method setting `iframe` to `true` like the following:

   ```
   $('#video-link').colorbox({
          iframe: true
   });
   ```

   Now our video loads into the Colorbox, but the Colorbox has no idea how large our video can be, so we can't see it.

6. We'll have to tell Colorbox how big we expect our video player to be. We'll do this by passing in key/value pairs for the `innerWidth` and `innerHeight`. We're using `innerWidth` and `innerHeight` rather than width and height in this case because we're passing in how large we want the video player (or content) to be, rather than how large we want the Colorbox to be:

   ```
   $('#video-link').colorbox({
          iframe: true,
          innerWidth: '640px',
          innerHeight: '480px'
   });
   ```

7. We can also use the Colorbox to create a way for users to easily view several videos. Let's go back into `index.html` and add a list of favorite videos to our page instead of just one link to a video. We'll use a `rel` attribute set to `favorites` for each one and provide a `title` attribute so our videos will display a caption underneath:

   ```
   <h3>Favorite Videos</h3>
   <ul>
          <li><a href="http://www.youtube.com/embed/
   itn8TwFCO4M?autoplay=1" rel="favorites" title="Louis CK and
   Everything is Amazing">Everything is Amazing</a></li>
          <li><a href="http://www.youtube.com/embed/
   UN0A6h9Wc5c?autoplay=1" rel="favorites" title="All This Beauty by
   The Weepies">All This Beauty</a></li>
          <li><a href="http://www.youtube.com/embed/ZWtZA-
   ZmOAM?autoplay=1" rel="favorites" title="ABC's That's
   Incredible">That's Incredible</a></li>
   </ul>
   ```

8. The only update we have to make to our JavaScript in `scripts.js` is to update the selector. Instead of selecting one single link by ID, we're going to select our set of favorites links by their `rel` attribute:

   ```
   $('a[rel="favorites"]').colorbox({
          iframe:true,
   ```

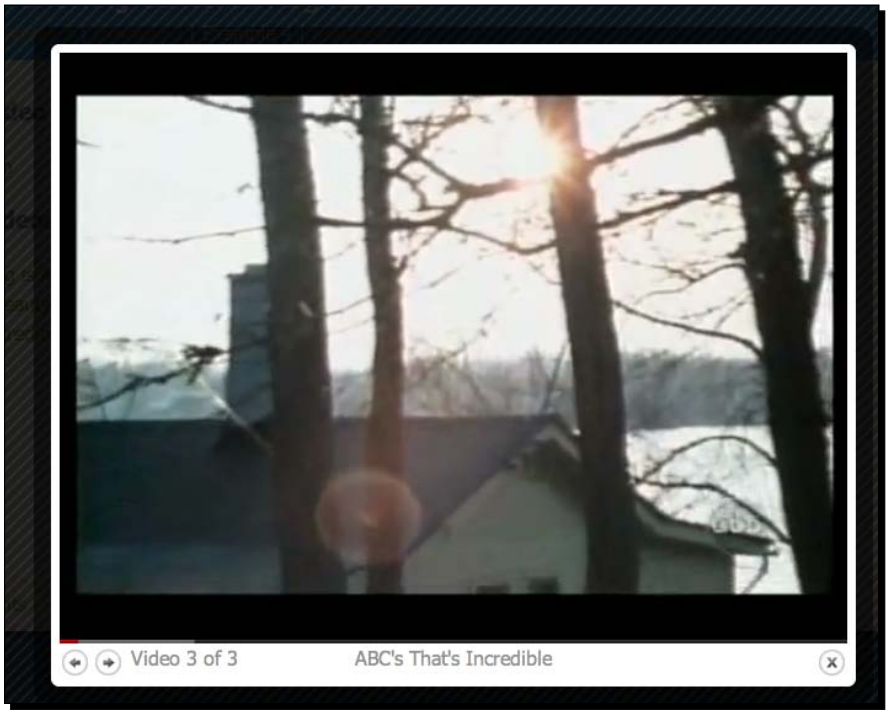
```
        innerWidth:'640px',
        innerHeight: '480px'
})
```

If you view the page in the browser, you'll see that you have a caption under the video and next and previous buttons that allow you to navigate between the videos without closing the Colorbox.

9. The only thing that's a bit awkward is that our pagination indicator says Image 1 of 3 when we're showing videos, not images. Luckily, Colorbox provides a way for us to customize this text with the `current` key:

```
$('a[rel="favorites"]').colorbox({
        iframe:true,
        innerWidth:'640px',
        innerHeight: '480px',
        current: 'Video {current} of {total}'
})
```

Now, our pagination indicator correctly reads Video 1 of 3. Our site visitors can easily move from video to video without having to close the Colorbox and each video displays a caption:



What just happened?

We learned how to create both a stand-alone video player and a multiple video player inside a Colorbox. We learned how to pass in key/value pairs to tell the Colorbox to load in external content in an `iframe`, working around cross-domain Ajax restrictions. We also learned how to modify the pagination indicator text to fit our current content type. We used the `innerWidth` and `innerHeight` keys to set the video player's size.

One-page web gallery

Next up, we'll take a look at how we can create a single-page web gallery to show off your favorite sites or all the incredible sites you've designed yourself. Note that this example makes use of Ajax, so you'll either have to upload your pages to a web server or create a web server on your own computer to see it in action.

Time for action – creating a one-page web gallery

Follow these steps to create a one-page web gallery:

1. We'll get started by setting up a basic HTML file and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*. Inside the body of our HTML document, we'll create a list of links to the sites we want to include in our design gallery:

   ```
   <h3>One-Page Web Design Gallery</h3>
   <ul>
          <li><a href="http://packtpub.com" rel="gallery">Packt
   Publishing</a></li>
          <li><a href="http://nataliemac.com"
   rel="gallery">NatalieMac</a></li>
          <li><a href="http://google.com" rel="gallery">Google</a></
   li>
   </ul>
   ```

   Note that I've added a `rel` attribute equal to `gallery` to each link.

2. Now, just as with the other Colorbox examples, choose a style and attach the stylesheet in the header of the document, make all the necessary images available to your page, update the path to the images in the CSS if necessary, and attach the Colorbox plugin at the bottom of the page.

3. Next, we'll open our `scripts.js` file and add our document ready statement:

```
$(document).ready(function(){

});
```

4. Next, we'll select all links with the `rel` attribute equal to `gallery` and call the `colorbox()` method:

```
$(document).ready(function(){
       $('a[rel="gallery"]').colorbox();
});
```

5. Just as we did with the video example, we'll set the `iframe` key to `true` since we're loading in content from other domains. I'm also going to set the `width` and `height` of the ColorBox to `90%`, so that it takes up nearly the entire browser window. I'm also going to adjust the pagination indicator text to read `Web Site` instead of `Image`:

```
$('a[rel="gallery"]').colorbox({
       iframe: true,
       width: '90%',
       height: '90%',
       current: 'Web Site {current} of {total}'
});
```

Now, if you refresh the page in the browser, you can see that clicking one of the links opens a Colorbox and loads that website into the Colorbox. A site visitor can interact with the loaded website just as they would if they had loaded it into a separate browser window, browsing through pages, and so on. When finished with one site, they can click the next arrow to visit the next website in the list and then hit the *Esc* key on the keyboard or click the close button or anywhere outside the Colorbox to close the Colorbox when they're finished.

Note that it is possible for website owners to block your ability to load their sites into an `iframe`. If you have set up a local server using MAMP or WAMP, then you might notice that the Google example won't load into your page. It will, however, load if you upload your code to an external server. Be sure to test all the sites you want to use in your web gallery to ensure that they work as expected.

What just happened?

We used much of what we learned creating a Colorbox video player to display external websites inside a Colorbox. This allows our site visitor to browse a collection of websites without ever leaving our page. We once again told Colorbox to load our content into an `iframe` to work around cross-domain Ajax restrictions. We customized the pagination indicator text, and set a width and height for our Colorbox.

Summary

We've looked at several uses for the adaptable and flexible Colorbox plugin, which can be used to display any kind of content in a lightbox. It can be used to create browsable image galleries, give access to forms and video players without cluttering up the page with clunky UI elements, and even to create a browsable website gallery. The ColorBox plugin is completely styled with CSS, making it possible for the lightbox to have any appearance you can dream up. The plugin even includes vector image assets that can be used as a starting point for creating your own lightbox design. The behavior of the lightbox can be modified by passing a series of key/value pairs to the `colorbox()` method, making the Colorbox plugin suitable for any possible lightbox use.

Next up, we'll take a look at another common website task: creating slideshows.

9
Creating Slideshows

Traditionally created in Flash, slideshows are a great way to show off photos, products, illustrations, portfolios, and more. Hands-down, creating slideshows is one of the most common tasks for jQuery developers. In this chapter we'll take a look at how to create a simple slideshow from scratch, and then we'll explore three powerful plugins that create gorgeous, dynamic, and full-featured slideshows.

In this chapter, we'll cover the following:

- How to plan a slideshow
- How to write a simple crossfading slideshow from scratch
- How to use the CrossSlide plugin to create a panning and zooming slideshow
- How to use the Nivo Slider plugin to create a slideshow with fun transition effects
- How to use the Galleriffic plugin to create a thumbnail slideshow

Planning a slideshow

There are a few things to consider when you're preparing to build a jQuery slideshow. They are as follows:

- First, you have to decide what will be the experience for users who have JavaScript disabled. The priority of the various pieces of content in the slideshow should be your guide. If the slideshow is simply featuring bits of content available elsewhere on the site, then it should be sufficient to simply show one photo or slide. If the slideshow is the only way to access the content, then you'll have to be sure to make that content available for users without JavaScript enabled. We'll take a look at both strategies in the various examples in this chapter.

- Second, you have to determine if all items in your slideshow will be of the same size or of different sizes. For obvious reasons, it's easiest to handle items that are all the same size and aspect ratio, but sometimes, it's impractical or impossible to identically size all items. I'll cover which slideshows are ideal for same-sized content and which are ideal for variable-sized content as we go along.
- Next, you need to consider if your site visitors need to have any kind of control over the slideshow. Sometimes, it's handy to simply have your images on automatic rotation. Other times, it's helpful to allow site visitors to pause the slideshow, or manually move forward and backward through the slides. I'll tell you how much control each of these slideshow approaches offers your site visitors.

Simple crossfade slideshow

In this section, you'll learn how to build a simple crossfade slideshow. This type of slideshow is ideal for identically-sized images and can be displayed as a single image when JavaScript is disabled. Finally, this type of slideshow offers no control over the slideshow to your site visitors. They cannot pause the slideshow or manually move through the slides.

Time for action – creating a simple crossfade slideshow

Follow these steps to create a simple crossfading slideshow:

1. We'll get started by creating a basic HTML document and associated files and folders just like we did in *Chapter 1*, *Designer, Meet jQuery*. In the body of the HTML document, include a list of images. Each list item will contain an image, which can optionally be wrapped in a link. Here's a sample of my image list:

```
<ul id="crossfade">
  <li>
    <a href="http://en.wikipedia.org/wiki/Agua_Azul"><img
src="images/600/AguaAzul.jpg" alt="Agua Azul"/></a>
  </li>
  <li>
    <a href="http://en.wikipedia.org/wiki/Burney_Falls"><img
src="images/600/BurneyFalls.jpg" alt="Burney Falls"/></a>
  </li>
  <li>
    <a href="http://en.wikipedia.org/wiki/Venezuala"><img
src="images/600/Cachoeira_do_Pacheco.jpg" alt="Cachoeira do
Pacheco"/></a>
  </li>
</ul>
```

2. Next, we'll write a few lines of CSS to style the slideshow. A slideshow shows just one image at a time and the easiest way to show only one image is to stack the images up on top of one another. If the site visitor has JavaScript disabled, they'll just see the last slide in the list:

   ```
   #crossfade  { position:relative;margin:0;padding:0;list-style-type
   :none;width:600px;height:400px;overflow:hidden; }
   #crossfade li  { position:absolute;width:600px;height:400px; }
   ```

   If you view the page in a browser, you'll see that the last item in the slideshow is visible, but none of the other items are—they are all stacked beneath the last item. This is what our experience will be for site visitors with JavaScript disabled.

3. Next, open up `scripts.js` and we'll get started writing our JavaScript code. This script will be a little bit different than scripts that we've set up before. Instead of something happening just once when the document loads or when a site visitor clicks a link, we actually want to set up a function that will happen on a timed interval. For example, if we want each slide of our slideshow to be visible for three seconds, we'll have to set up a function to switch slides, that gets called every three seconds.

   We've already got our slides stacked up on top of one another on the page with the last item on top. Think about how you handle a stack of photographs. You view the photograph on top, and then move it to the bottom of the stack to view the second photo. Then you move the second photo to the bottom to view the third photo and so on. We're going to apply the same principle to our slideshow.

   Inside `scripts.js`, create a function called `slideshow`. This is the function that we'll call every three seconds when we want to switch photos.

   ```
     function slideshow() {
     }
   ```

4. The first thing we need to do inside our function is select the first photo in the stack.

   ```
   function slideshow() {
     $('#crossfade li:first')
   }
   ```

5. Now we've got the first photo in the stack, we just need to move it to the bottom of the stack to make the next photo visible. We can do that by using jQuery's `appendTo()` method. This will remove the first photo from the beginning of the list and append it to the end of the list.

   ```
   function slideshow() {
     $('#crossfade li:first').appendTo('#crossfade');
   }
   ```

6. Our photo-flipping function is ready. Now all we have to do is some initial setup as soon as our page loads. Then we'll set up a call to our photo-flipping function every three seconds. We'll call the `ready()` method on the document.

```
$(document).ready(function(){
  // Document setup code will go here
});

function slideshow() {
  $('#crossfade li:first').appendTo('#crossfade');
}
```

7. As soon as our document is ready, we want to prepare our slideshow. We'll start by selecting all the photos in the slideshow.

```
$(document).ready(function(){
  $('#crossfade li')
});
```

8. Next, we want to hide all the photos in the slideshow.

```
$(document).ready(function(){
  $('#crossfade li').hide();
});
```

9. Then, we'll filter that list of photos to get just the first one.

```
$(document).ready(function(){
  $('#crossfade li').hide().filter(':first');
});
```

10. And finally, we'll make that first photo visible. All other photos will remain hidden.

```
$(document).ready(function(){
  $('#crossfade li').hide().filter(':first').show();
});
```

11. At this point, if you refresh the page in the browser, you'll see that the last slide visible without JavaScript enabled is now hidden and the first slide in the list is now visible instead. Now, all that's left to do is to call our photo-flipping function every three seconds. To do this, we'll use a JavaScript method called `setInterval()`. This allows us to call a function at a regular interval. We pass two values to `setInterval`: the name of the function to be called and the number of milliseconds that should elapse between calls to the function. For example, to call my slideshow function every three seconds (or 3000 milliseconds), I'd write:

```
$(document).ready(function(){
  $('#crossfade li').hide().filter(':first').show();
  setInterval(slideshow, 3000);
});
```

12. Now, we're calling our photo-flipping function every three seconds, so you'd expect that if you refresh the page in the browser then you'd see the photos changing every three seconds, but that doesn't appear to be the case. Reviewing the code, it's easy to see what's gone wrong—even though the actual order of the stack of photos is changing every three seconds, all the photos except the first one are invisible. Whether the first photo is on top or not, it's the only photo visible, so it appears that our slideshow isn't changing. We'll have to go back to our `slideshow` function and modify it to make the current photo invisible and make the next photo in the stack visible. Since we want the photos to switch with a nice, slow crossfading effect, we'll call the `fadeOut()` method to fade the first photo to transparent, and we'll pass `slow` to that method to ensure it takes its time:

```
function slideshow() {
  $('#crossfade li:first').fadeOut('slow').appendTo('#crossfade');
}
```

13. Now, we need to move to the next photo in the list which is currently invisible and make it opaque. We're going to use the `next()` method to get the next item in the list and then call the `fadeIn()` method to make it appear. Once again, since we want a slow effect, we'll pass `slow` to the `fadeIn()` method:

```
function slideshow() {
  $('#crossfade li:first').fadeOut('slow').next().fadeIn('slow').
appendTo('#crossfade');
}
```

14. Finally, we've gotten ourselves into a little bit of trouble with our chaining of jQuery methods. We started with the first photo in the stack, faded it out, then moved to the second photo in the stack, and faded it in. However, when we call the `appendTo()` method, we're appending the second photo in the stack to the end—we're moving the second photo in the stack to the bottom instead of the first one. Luckily, jQuery provides a method for us to return to our original selection—the `end()` method. We can call the `end()` method after fading in the second photo to make sure that it's the first photo that's getting appended to the bottom of the photo stack:

```
function slideshow() {
  $('#crossfade li:first').fadeOut('slow').next().fadeIn('slow').
end().appendTo('#crossfade');
}
```

What just happened?

If you refresh the page in the browser, you'll see that you've got a nice crossfading slideshow. As one photo fades out, the next photo fades in, smoothly transitioning between each photo. Since we're constantly moving the top photo in the stack to the bottom, we'll never reach the end of the slideshow, just as you can continuously flip through a stack of photos.

We set up a slideshow function that selected the first photo in the stack, faded it out, and moved it to the bottom of the stack. Simultaneously, we're finding the second photo in the stack and fading it in. We used the power of jQuery chaining to accomplish all of that in one line of code.

We set up an interval of three seconds and called our photo-flipping function at the end of each three second interval.

Finally, we did a bit of set-up work as soon as the document is loaded—hiding all the photos and then making the first one visible. This will ensure that the photos are always displayed in order in our slideshow.

Next up, let's take a look at another plugin with some fancy transition effects.

Nivo Slider

In this section, we'll take a look at how to put the Nivo Slider plugin from Dev 7 Studios to good use. Nivo Slider provides for some eye-popping transition effects between photos and offers lots of configuration options. Nivo Slider is ideal for photos that are all identically-sized and it's easy to display a single photo in place of the slideshow for users with JavaScript disabled. Site visitors have the ability to manually advance forward and backward through the slideshow and the slideshow pauses when the mouse is moved over it.

Nivo Slider is a little different than most of the plugins we'll take a look at in this book. The plugin itself is open-source under the MIT license (`http://nivo.dev7studios.com/license/`) and is free to download and use. There are also paid versions of the plugin available for WordPress users that include support, automatic updates, and permission to include the plugin with premium WordPress themes. The slideshow we create in this section is using the free, open-source version of the plugin.

Time for action – creating a Nivo Slider slideshow

Follow these steps to create an image slideshow with fancy transitions:

1. We'll get started by setting up a basic HTML file and associated files and folders just like we did in *Chapter 1*, *Designer, Meet jQuery*. In the body of the HTML document, Nivo Slider simply requires a set of images inside a container `<div>`.

   We can optionally wrap each image in a link if we want each slide of our slideshow to link off to another page or web location, but it's not required. Nivo will work fine with unlinked images as well. The `title` attribute of the `<img>` tag is used to display captions for the slideshow.

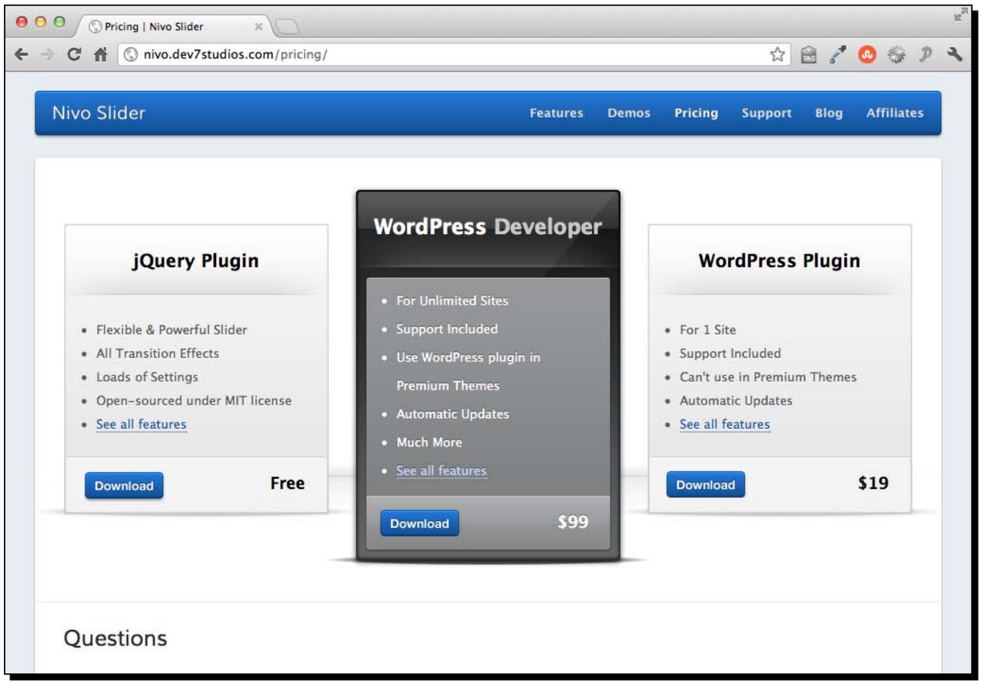
```
<div id="slideshow">
  <a href="http://en.wikipedia.org/wiki/Agua_Azul"><img
src="images/600/AguaAzul.jpg" alt="Agua Azul" title="Agua Azul,
Mexico"/></a>
  <a href="http://en.wikipedia.org/wiki/Burney_Falls"><img
src="images/600/BurneyFalls.jpg" alt="Burney Falls" title="Burney
Falls, California, USA"/></a>
  <a href="http://en.wikipedia.org/wiki/Venezuala"><img
src="images/600/Cachoeira_do_Pacheco.jpg" alt="Cachoeira do
Pacheco" title="Cachoeira do Pacheco, Venezuela"/></a>
  <a href="http://en.wikipedia.org/wiki/Deer_Leap_Falls"><img
src="images/600/Deer_Leap_Falls.jpg" alt="Deer Leap Falls"
title="Deer Leap Falls, Pennsylvania, USA"/></a>
  <a href="http://en.wikipedia.org/wiki/Fulmer_Falls"><img
src="images/600/Fulmer_Falls.jpg" alt="Fulmer Falls" title="Fulmer
Falls, Pennsylvania, USA"/></a>
  <a href="http://en.wikipedia.org/wiki/Hopetoun_Falls"><img
src="images/600/Hopetoun_Falls.jpg" alt="Hopetoun Falls"
title="Hopetoun Falls, Victoria, Australia"/></a>
  <a href="http://en.wikipedia.org/wiki/Ohiopyle_State_Park"><img
src="images/600/Jonathans_Run.jpg" alt="Jonathan's Run"
title="Jonathan's Run, Pennsylvania, USA"/></a>
  <a href="http://en.wikipedia.org/wiki/Kjosfossen"><img
src="images/600/Kjosfossen.jpg" alt="Kjosfossen"
title="Kjosfossen, Norway"/></a>
  <a href="http://en.wikipedia.org/wiki/Krimml_Waterfalls"><img
src="images/600/KrimmlFalls.jpg" alt="Krimml Falls" title="Krimml
Falls, Salzburgerland, Austria"/></a>
  <a href="http://en.wikipedia.org/wiki/Madhabkunda"><img
src="images/600/Madhabkunda_Falls.jpg" alt="Madhabkunda Falls"
title="Madhabkunda Falls, Bangladesh"/></a>
  <a href="http://en.wikipedia.org/wiki/Manavgat_Waterfall"><img
src="images/600/Manavgat.jpg" alt="Manavgat Waterfall"
title="Manavgat Waterfall, Turkey"/></a>
```

```
  <a href="http://en.wikipedia.org/wiki/Niagra_Falls"><img
src="images/600/Niagara_Falls.jpg" alt="Niagara Falls"
title="Niagara Falls, USA and Canada"/></a>
  <a href="http://en.wikipedia.org/wiki/British_Columbia"><img
src="images/600/Nymph_Falls.jpg" alt="Nymph Falls" title="Nymph
Falls, British Columbia, Canada"/></a>
</div>
```

2. Next, we'll add some CSS that will stack up the images on top of one another and set a fixed width and height for our slideshow:

```
#slideshow { position:relative;width:600px;height:400px; }
#slideshow img { position:absolute;top:0;left:0; }
```

3. Now, head over to `http://nivo.dev7studios.com/pricing/` to download the Nivo Slider plugin. You'll find the **Download** link in the left box that's labeled **jQuery plugin**.



jQuery Plugin
- Flexible & Powerful Slider
- All Transition Effects
- Loads of Settings
- Open-sourced under MIT license
- See all features

Free

WordPress Developer
- For Unlimited Sites
- Support Included
- Use WordPress plugin in Premium Themes
- Automatic Updates
- Much More
- See all features

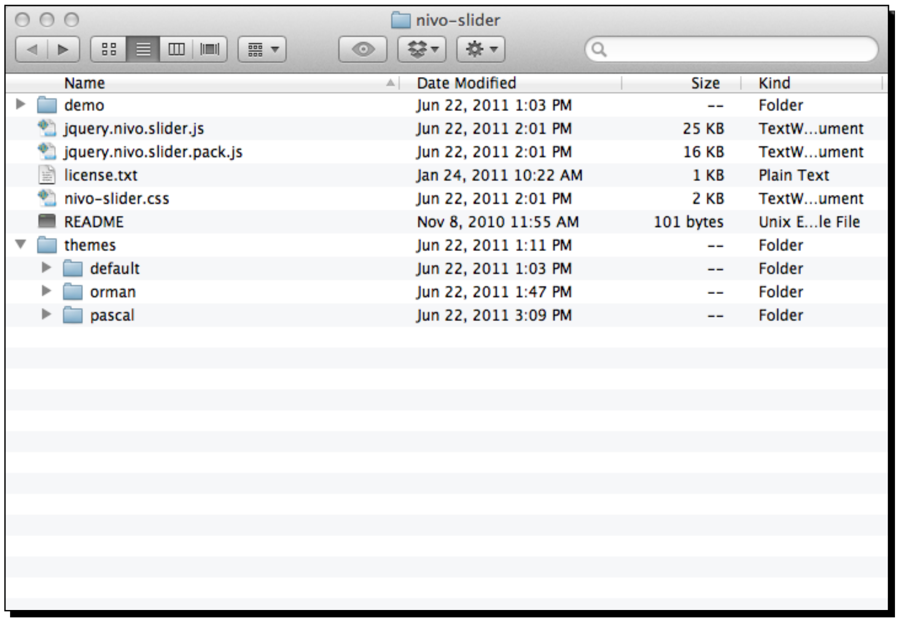
$99

WordPress Plugin
- For 1 Site
- Support Included
- Can't use in Premium Themes
- Automatic Updates
- See all features

$19

Click the **Download** link and save the zip file to your computer.

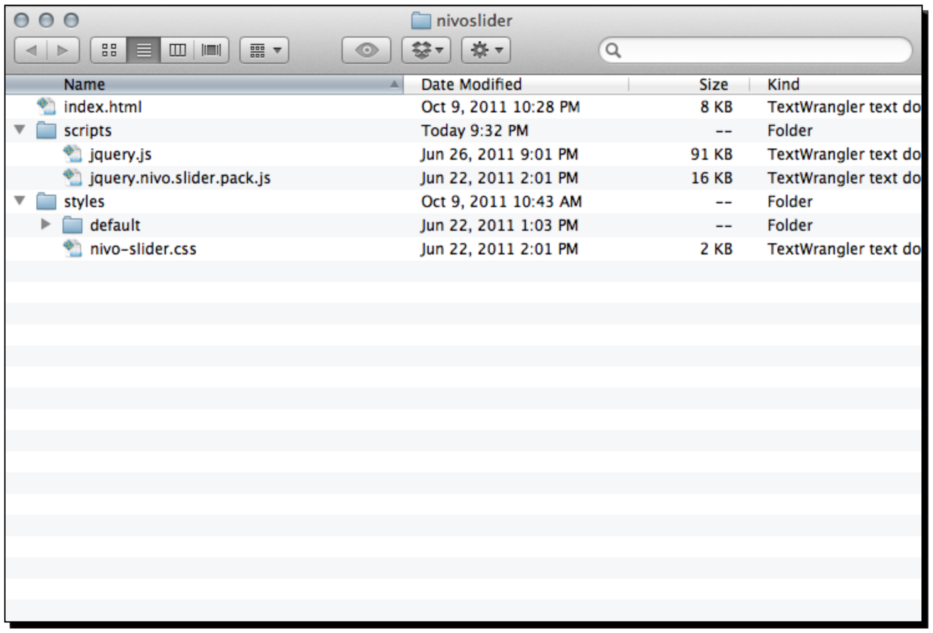
4. Unzip the folder and take a look inside.



Name	Date Modified	Size	Kind
demo	Jun 22, 2011 1:03 PM	--	Folder
jquery.nivo.slider.js	Jun 22, 2011 2:01 PM	25 KB	TextW...ument
jquery.nivo.slider.pack.js	Jun 22, 2011 2:01 PM	16 KB	TextW...ument
license.txt	Jan 24, 2011 10:22 AM	1 KB	Plain Text
nivo-slider.css	Jun 22, 2011 2:01 PM	2 KB	TextW...ument
README	Nov 8, 2010 11:55 AM	101 bytes	Unix E...le File
themes	Jun 22, 2011 1:11 PM	--	Folder
default	Jun 22, 2011 1:03 PM	--	Folder
orman	Jun 22, 2011 1:47 PM	--	Folder
pascal	Jun 22, 2011 3:09 PM	--	Folder

There's a **demo** folder that contains a sample HTML file along with images, scripts, and styles. There are two versions of the plugin—the source version and a packed and minified version. There's a copy of the license, which is shorter and simpler than you might expect, so feel free to take a look at it. There's a CSS file, and then there's a **themes** folder that contains three other folders: **default**, **orman**, and **pascal**. These are three sample themes included with the plugin. You choose one of these sample themes, create your own, or modify one of the sample themes to suit your tastes.

5. Let's get the necessary files copied over and ready to use. First, copy `nivo-slider.css` to your own `styles` folder. Select one of the themes and copy the entire folder to your own `styles` folder as well. Then copy `jquery.nivo.slider.pack.js` to your own `scripts` folder alongside jQuery. Your setup should look like the following image:



Name	Date Modified	Size	Kind
index.html	Oct 9, 2011 10:28 PM	8 KB	TextWrangler text do
▼ scripts	Today 9:32 PM	--	Folder
jquery.js	Jun 26, 2011 9:01 PM	91 KB	TextWrangler text do
jquery.nivo.slider.pack.js	Jun 22, 2011 2:01 PM	16 KB	TextWrangler text do
▼ styles	Oct 9, 2011 10:43 AM	--	Folder
▶ default	Jun 22, 2011 1:03 PM	--	Folder
nivo-slider.css	Jun 22, 2011 2:01 PM	2 KB	TextWrangler text do

6. Next, we'll get our HTML file set up to use Nivo Slider. In the `<head>` section of the document, include the `nivo-slider.css` file along with the `CSS` file for the theme you've selected, before your `styles.css` file:

```
<head>
  <title>Chapter 9: Creating Slideshows</title>
  <link rel="stylesheet" href="styles/nivo-slider.css"/>
  <link rel="stylesheet" href="styles/default/default.css"/>
  <link rel="stylehseet" href="styles/styles.css"/>
</head>
```

7. At the bottom of the HTML document, just below the closing `</body>` tag, insert the `<script>` tag to include the Nivo Slider plugin, between jQuery and your `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.nivo.slider.pack.js"></script>
<script src="scripts/scripts.js"></script>
</body>
```

8. Open `scripts.js` and call the `ready()` method on the document so that our slideshow will start as soon as the page is loaded in the browser window:

```
$(document).ready(function(){
  //Nivo Slider code will go here
});
```

9. Next, we'll select the container element for our slideshow:

```
$(document).ready(function(){
  $('#slideshow');
});
```

10. And finally, we'll call the `nivoSlider()` method:

```
$(document).ready(function(){
  $('#slideshow').nivoSlider();
});
```

    Now if you view the page in the browser, you'll see our slideshow has been created. The default setting for the transition effect is to use a random different effect for each transition, so if you watch for a few minutes, you'll get a good idea of the different types of transition effects Nivo Slider includes.

    You'll also notice that the value we've included as the `title` attribute for each image is being displayed as the caption for each image.

11. Now let's take advantage of some of the customization options available with the Nivo Slider plugin. The documentation for our options is available at `http://nivo.dev7studios.com/support/jquery-plugin-usage/`

    You'll find a list of the available transitions at the bottom of the documentation page. My personal favorite transition is called `boxRain`. Let's set that up to be the only transition effect used. We'll customize the Nivo Slider plugin by passing a set of key/value pairs to the `nivoSlider()` method inside a set of curly brackets:

```
$(document).ready(function(){
  $('#slideshow').nivoSlider({
    effect: 'boxRain'
  });
});
```

12. We can specify the number of rows and columns the box animation should include. By default, there are eight columns and four rows, but let's increase that so that the `boxRain` transition uses more (smaller) boxes:

```
$(document).ready(function(){
  $('#slideshow').nivoSlider({
    effect:  'boxRain',
    boxCols:  10,
    boxRows:  5
  });
});
```

13. We can also customize the animation speed and the amount of time each slide is shown:

```
$(document).ready(function(){
  $('#slideshow').nivoSlider({
    effect:  'boxRain',
    boxCols:  10,
    boxRows:  5,
    animSpeed:  800,
    pauseTime:  4000
  });
});
```

I've set the `animSpeed` to 800 milliseconds so that the `boxRain` transition effect will take 800 milliseconds to complete. I've also set the `pauseTime` to 4000, so that each image in the slideshow is visible for 4000 milliseconds or four seconds.

What just happened?

We set up the Nivo Slider plugin to showcase a slideshow with impressive transition effects. We learned how to set up the HTML document appropriately, how to call the `nivoSlider()` method and how to customize some of the slideshow settings.

Have a go hero – customize the slideshow

In addition to the customization options we used, there are several other configuration options available for the slideshow including the ability to show or hide next/previous buttons, choices for setting up the pagination display or whether or not to show it at all, and lots of callback functions for writing custom functionality for the slideshow. On top of all that, you can completely customize the CSS and images used to create the slideshow so that it looks any way you'd like.

Try your hand at customizing a slideshow to match any design you'd like and experiment with the other customization options the Nivo Slider makes available. Create a custom slideshow of your own design.

Next up, we'll take a look at creating a thumbnail photo gallery.

Galleriffic slideshow

The Galleriffic slideshow by Trent Foley allows you to turn a list of links to full-size photos into a photo slideshow. The approach is a bit different than the other galleries we've seen so far, where the focus has been on inserting the full-size photos in the document and then animating them into a slideshow. Galleriffic instead takes a list of links to the full-size photos and turns that into a slideshow. The links remain on the page as one way to navigate through the slideshow.

The Galleriffic slideshow can be used with a set of photos that vary somewhat in size and aspect ratio, but if the difference between different photos is too great, getting the CSS set up to handle the slideshow gracefully will be quite a challenge. The Galleriffic slideshow makes it easy for your site visitor to manually navigate to any photo in the slideshow and also provides next, previous, and a play/pause button for the slideshow. For site visitors with JavaScript disabled, a list of links will be provided that will link them to the full-size versions of the photos.

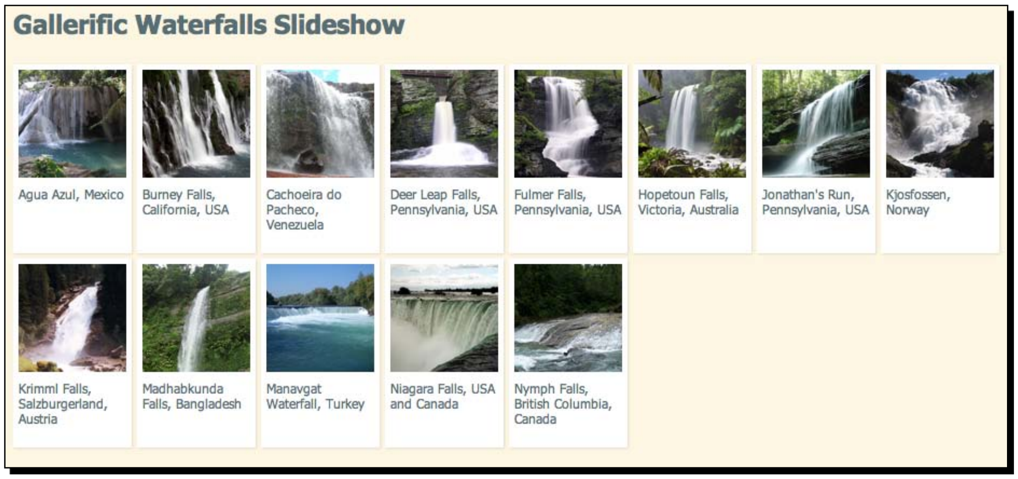
We're also going to explore a simple technique that you can use to apply different CSS to the page depending on whether or not JavaScript is enabled. This technique can be applied in a variety of circumstances to give you a bit more control over how your content is presented for site visitors when they have JavaScript disabled.

Time for action – creating a Galleriffic slideshow

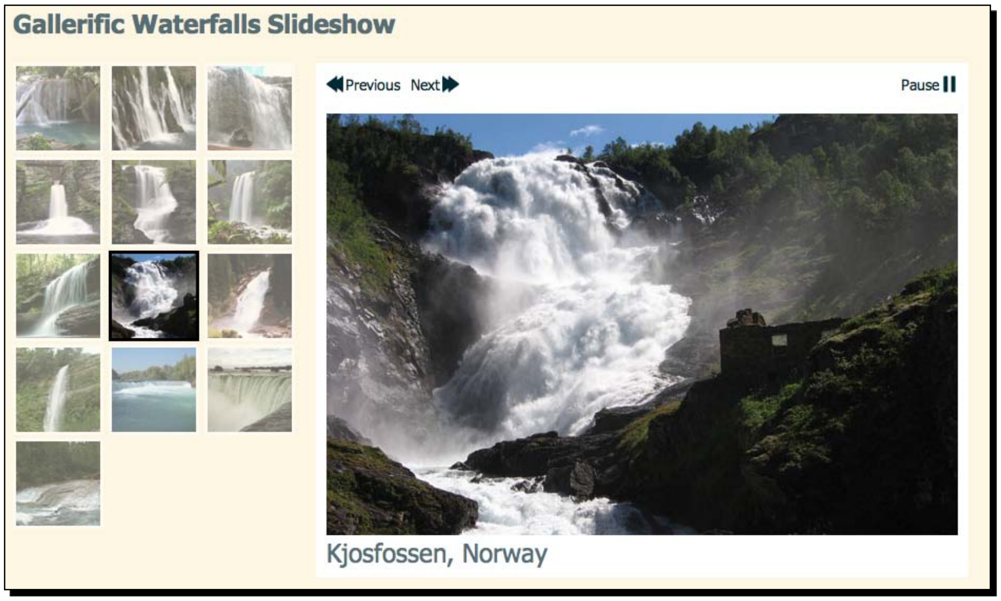
Follow these steps to create a slideshow using the Galleriffic plugin:

1. First up, we're going to make some extra effort to plan out how the slideshow will appear for site visitors with and without JavaScript enabled. If the site visitor doesn't have JavaScript, we'll present them with a grid of thumbnails with captions beneath. Clicking on a thumbnail will show them the full-size version of the photo.

The page will look like the following screenshot:


Gallerific Waterfalls Slideshow
Agua Azul, Mexico
Burney Falls, California, USA
Cachoeira do Pacheco, Venezuela
Deer Leap Falls, Pennsylvania, USA
Fulmer Falls, Pennsylvania, USA
Hopetoun Falls, Victoria, Australia
Jonathan's Run, Pennsylvania, USA
Kjosfossen, Norway
Krimml Falls, Salzburgerland, Austria
Madhabkunda Falls, Bangladesh
Manavgat Waterfall, Turkey
Niagara Falls, USA and Canada
Nymph Falls, British Columbia, Canada

For users with JavaScript, though, I want to show a smaller grid of thumbnails beside a main slideshow area like in the following screenshot:



The captions aren't important in the case of thumbnails because they'll be displayed below the slideshow rather than below the photos.

2. Keeping in mind how we want the page to appear, we'll get started by setting up an HTML file and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*. Create a set of 100x100 thumbnails for each photo and store them in a `thumbs` folder inside your `images` folder. We'll use these thumbnails to create a list of links to the full-size photos in the body of the HTML document.

```
<ul class="thumbs">
  <li>
    <a class="thumb" title="Agua Azul, Mexico" href="images/600/
AguaAzul.jpg"><img src="images/thumbs/AguaAzul.png" alt="Agua
Azul"/></a>
    <div class="caption">Agua Azul, Mexico</div>
  </li>
  <li>
    <a class="thumb" title="Burney Falls, California, USA"
href="images/600/BurneyFalls.jpg"><img src="images/thumbs/
BurneyFalls.png" alt="Burney Falls"/></a>
    <div class="caption">Burney Falls, California, USA</div>
  </li>
  <li>
    <a class="thumb" title="Cachoeira do Pacheco, Venezuela"
href="images/600/Cachoeira_do_Pacheco.jpg"><img src="images/
thumbs/Cachoeira_do_Pacheco.png" alt="Cachoeira do Pacheco"/></a>
    <div class="caption">Cachoeira do Pacheco, Venezuela</div>
  </li>
  <li>
    <a class="thumb" title="Deer Leap Falls, Pennsylvania, USA"
href="images/600/Deer_Leap_Falls.jpg"><img src="images/thumbs/
Deer_Leap_Falls.png" alt="Deer Leap Falls"/></a>
    <div class="caption">Deer Leap Falls, Pennsylvania, USA</div>
  </li>
</ul>
```

We've included a `title` attribute on each link to ensure a tooltip will show when the mouse hovers over each thumbnail with this brief photo description. I've also included an `alt` attribute on each image tag so that site visitors unable to see the images for any reason will still have access to this description of the image.

Also inside each `<li>`, I've included a `<div>` with a class of `caption` that contains the caption that will appear beneath the thumbnails or beneath the photos in the slideshow.

This is enough HTML to get the non-JavaScript version of the slideshow set up, but the Galleriffic plugin requires a few more elements on the page.

3. We need to wrap our list of images in a `<div>` with an `id` of `thumbs` like the following:

```
<div id="thumbs">
  <ul class="thumbs">
    <li>
    ...
    </li>
  </ul>
</div>
```

4. We also need to add some empty elements to the page that will hold our slideshow, slideshow caption, and slideshow controls.

```
<div id="thumbs">...</div>
<div id="gallery">
  <div id="controls"></div>
  <div id="slideshow-container">
    <div id="loading"></div>
    <div id="slideshow"></div>
  </div>
  <div id="caption"></div>
</div>
```

   The exact position of these elements on the page is up to you—you can create whatever layout you'd like and put the various parts of the slideshow on the page wherever you'd like. For usability purposes, of course, the elements should all be relatively close together.

   Note that aside from the thumbs `div` that contains our list of thumbnails, the other elements we've added to the page are empty. These elements will only be used if the site visitor has JavaScript enabled, and all the content inside them will be automatically generated by the Galleriffic plugin. This makes them invisible unless they're being used.

5. Now, open your HTML file and find the opening `<body>` tag. Add a `class` of `jsOff`.

```
<body class="jsOff">
```

6. Next, we'll set up the CSS styles for the thumbnails. Open your `styles.css` file and add these styles:

   ```
   .thumbs    { margin:0;padding:0;line-height:normal; }
   .thumbs li  { display:inline-block;vertical-align:top;
     padding:0;list-style-type:none;margin:0; }

   .jsOff .thumbs li  { width:100px;margin-bottom:5px;background:#fff;
     border:5px solid #fff;box-shadow:1px 1px 2px rgba(0,0,0,0.1) }
   .jsOff .caption    { min-height:52px;font-size:12px;
     line-height:14px; }
   .jsOff #gallery    { display:none; }
   ```

   The CSS has two sections here. The selectors that simply start with `.thumbs` will apply to the thumbnails whether or not the site visitor has JavaScript enabled. The selectors that start with `.jsOff` will only apply to site visitors who do not have JavaScript enabled. This CSS creates the grid of thumbnails with captions beneath them.

   We've also selected the parent container for the slideshow and set it to not display at all for site visitors without JavaScript. Since it's a set of empty `<div>`s they shouldn't take up any space on the page anyway, but this is some extra insurance that these extra elements won't cause any problems for site visitors without JavaScript.

   The non-JavaScript version of the page is complete.

7. Next, we'll set up the page for users who do have JavaScript enabled. We'll get started by opening up the `scripts.js` file and inserting our document ready statement:

   ```
     $(document).ready(function(){
     // This code will run as soon as the page loads
     });
   ```

8. Next, we'll write a bit of code to remove that `jsOff` class from the body and replace it with a `jsOn` class.

   ```
   $(document).ready(function(){
     $('body').removeClass('jsOff').addClass('jsOn');
   });
   ```

   If a site visitor has JavaScript, the `jsOff` class will be removed from the body and replaced by a `jsOn` class.

9. Now, we can write some CSS to apply to the list of thumbnails for site visitors who do have JavaScript. Open your `styles.css` file and add these styles:

```
.jsOn .thumbs    { width:288px; }
.jsOn .thumbs li  { width:86px; }
.jsOn .thumbs img  { border:3px solid #fff;max-
width:80px;opacity:0.6; }
.jsOn #thumbs    { float:left; }
```

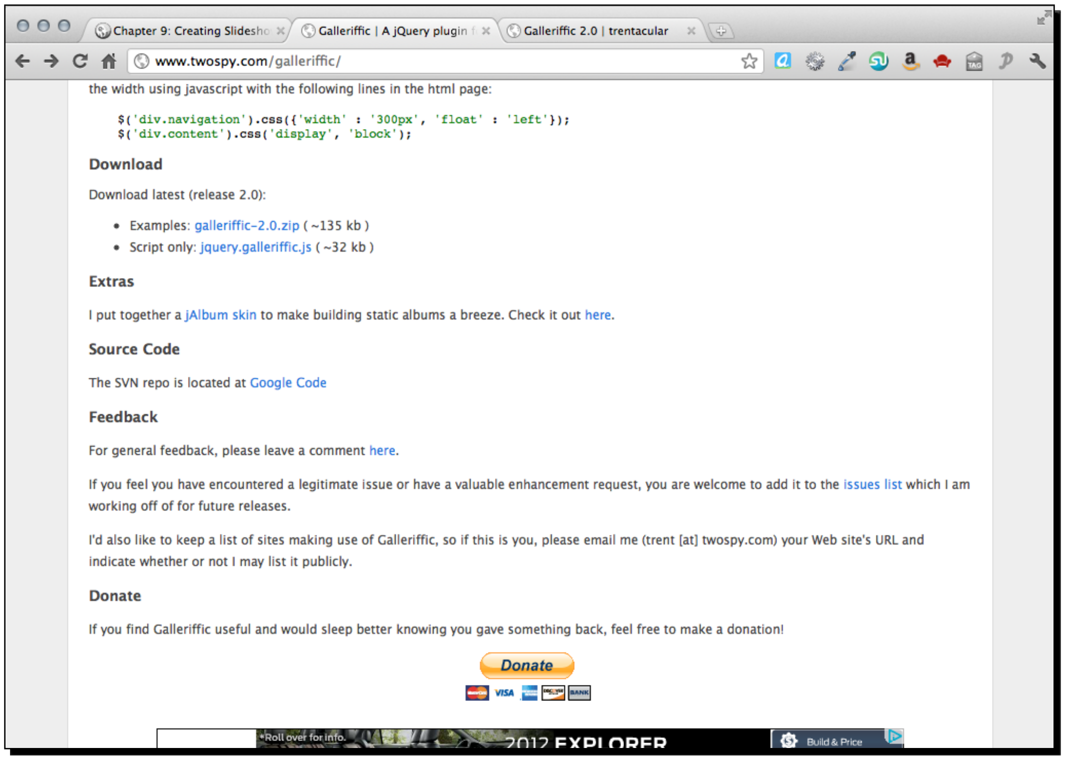
This CSS will only apply to site visitors with JavaScript enabled, since the `jsOn` class can only be applied to the `<body>` if JavaScript is available to do the work.

10. Now, we'll write some styles for the bits that make up the slideshow—the controls, the caption, and the slideshow area itself:

```
#gallery      { float:left;width:600px;position:relative;backgroun
d:#fff;padding:10px;margin-bottom:20px;line-height:18px; }
.ss-controls    { text-align:right;float:right;width:40%; }
.nav-controls    { float:left:width:40%; }
#controls a      { font-size:14px;color:#002B36;background:100%
0px no-repeat url(images/controls/sprite.png);padding-right:18px;
}
#controls a.pause  { background-position: 100% -18px; }
#controls a.prev  { background-position: 0 -36px;padding-
right:0;padding-left:18px;margin-right:10px; }
#controls a.next  { background-position: 100% -54px; }
.caption      { font-size:24px;padding:5px 0; }
.thumbs li.selected img  { border-color:#000;opacity:1; }
```

I've created a small sprite that contains images for **play**, **pause**, **previous**, and **next** that I'm applying to these controls.

11. Now that we're all set up to create an awesome slideshow, we just need our plugin code. Head over to `http://www.twospy.com/galleriffic/` where you'll find the documentation and downloads for the Galleriffic plugin. You'll have to scroll down the page nearly to the bottom to find the **Download** section.


Chapter 9: Creating Slidesho
Galleriffic | A jQuery plugin
Galleriffic 2.0 | trentacular
www.twospy.com/galleriffic/

the width using javascript with the following lines in the html page:

```
$('div.navigation').css({'width' : '300px', 'float' : 'left'});
$('div.content').css('display', 'block');
```

Download

Download latest (release 2.0):

- Examples: galleriffic-2.0.zip (~135 kb)
- Script only: jquery.galleriffic.js (~32 kb)

Extras

I put together a jAlbum skin to make building static albums a breeze. Check it out here.

Source Code

The SVN repo is located at Google Code

Feedback

For general feedback, please leave a comment here.

If you feel you have encountered a legitimate issue or have a valuable enhancement request, you are welcome to add it to the issues list which I am working off of for future releases.

I'd also like to keep a list of sites making use of Galleriffic, so if this is you, please email me (trent [at] twospy.com) your Web site's URL and indicate whether or not I may list it publicly.

Donate

If you find Galleriffic useful and would sleep better knowing you gave something back, feel free to make a donation!

You'll notice that you have two options for the download—you can get a ZIP file that includes some examples or just the plugin code by itself. Since we already know what we want the slideshow to look like, we'll grab just the plugin code. Clicking the link will open up the code itself in the browser window. Right click or select **File | Save As** from the browser's menu to save the file to your own `scripts` folder.

12. Now that we've got the plugin, we want to include it in our HTML page. Go down to the bottom of the your HTML page and insert the Galleriffic plugin between jQuery and your `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.galleriffic.js"></script>
<script src="scripts/scripts.js">
</script>
```

13. Next, we'll open `scripts.js` and select the container that's wrapped around our list of thumbnails and call the `galleriffic()` method just after our line of code that changes the class on the body:

```
$('body').removeClass('jsOff').addClass('jsOn');
$('#thumbs').galleriffic();
```

14. But if you view the page in the browser, you'll see that the slideshow isn't working. This is because the Galleriffic plugin requires a bit of configuration to run. We're going to pass a set of key/value pairs inside curly brackets to the `galleriffic()` method so that our slideshow will run. We basically have to tell the plugin where to show our slideshow, controls, and caption.

```
$('#thumbs').galleriffic({
  imageContainerSel:    '#slideshow',
  controlsContainerSel:  '#controls',
  captionContainerSel:  '#caption',
  loadingContainerSel:  '#loading',
  autoStart:        true
});
```

The `<div>` with the `id` of `slideshow` is where we're going to show the full-size images. Controls will be shown in the `div` with the `id` of `controls`. `<div id="caption">` will display the caption and the `div` we created with the `id` of `loading` will display a loading animation while the slideshow initializes. I've also set `autoStart` to `true` so that the slideshow will start playing automatically.

Now if you refresh the page in the browser, you'll see the slideshow in action. **Next** and **previous** buttons allow you to flip through and a **play**/**pause** button gives you control over the slideshow.

What just happened?

We set up our page with a display of image thumbnails optimized for site visitors with JavaScript disabled. Then we used a line of JavaScript to change the body class so that we could apply different styles for site visitors who had JavaScript enabled. We set up CSS to display our slideshow and called the `galleriffic()` method to animate the slideshow. Site visitors can manually move back and forward through the photos, can click a thumbnail to load the corresponding full-size photo into the slideshow area, and can pause the slideshow at any point.

The CrossSlide plugin

The CrossSlide plugin, by Tobia Conforto, makes it possible to not just fade images in and out, but to also animate panning and zooming. This plugin is ideal if you have a variety of different image sizes. For best results, the only requirement is that all images are at least as large as the slideshow viewing area. Images larger than the slideshow viewing area will be cropped. For example, if the slideshow is 600 pixels wide by 400 pixels tall, then all images used in the slideshow should be at least 600 pixels wide and 400 pixels tall.

With JavaScript disabled, the CrossSlide plugin displays whatever content you've placed into the slideshow as a placeholder. This could be a single image, or it could be an image accompanied by text, or any other sort of HTML content you'd like. The plugin will then remove this placeholder content when the page loads and replace it with the slideshow.

It is possible to provide buttons that allow site visitors to stop and restart the slideshow. However, site visitors cannot manually advance through the various slides.

Before we dive in, I do want to give a fair warning that you'll find the CrossSlide plugin a bit less designer-friendly than some of the other plugins we've seen. A panning and zooming slideshow is a complex task, and the plugin can only do so much to take that complexity out of your hands. That said, I'm sure if you take your time and exercise a little patience, you'll be able to get it figured out.

Time for action – building a CrossSlide slideshow

Follow these steps to set up a CrossSlide slideshow:

1. To get started, we'll set up a simple HTML document and associated files and folders just like we did in *Chapter 1, Designer, Meet jQuery*. The body of the HTML document will contain a container for your slideshow. Inside the container, place any content you'd like to display for users with JavaScript disabled.

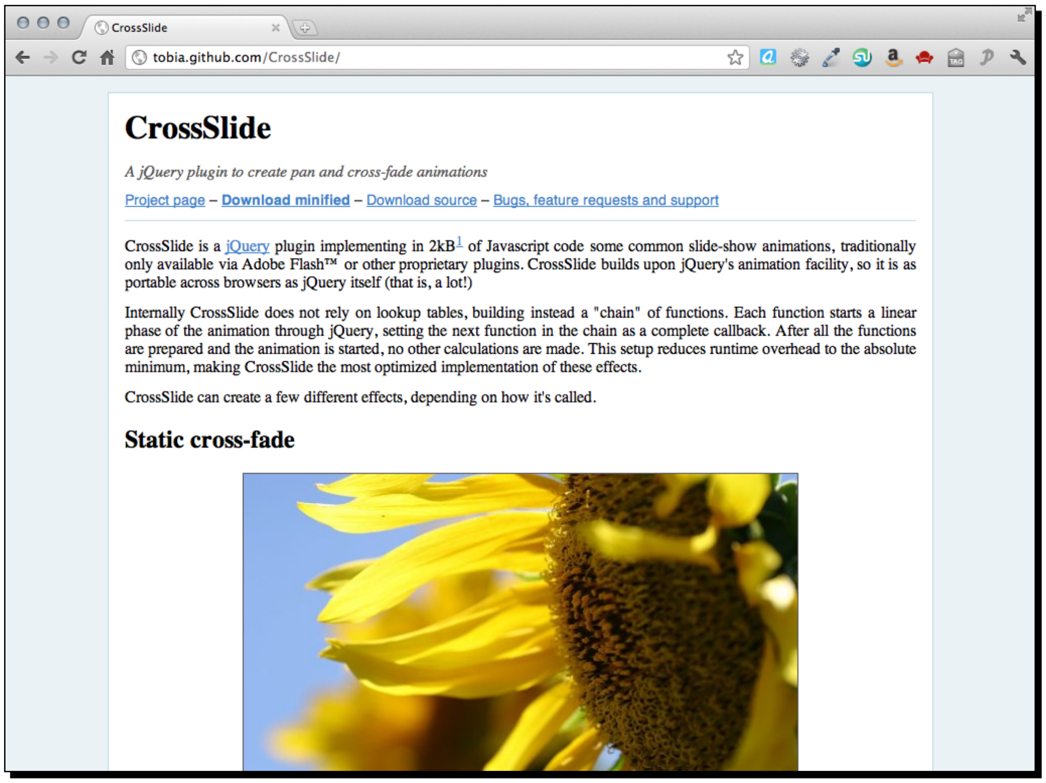
   ```
   <div id="slideshow">
     <img src="images/600/AguaAzul.jpg" alt="Agua Azul"/>
   </div>
   ```

   I'm going to simply show the first photo from the slideshow for users with JavaScript disabled. I've given my container `<div>` an `id` of `slideshow`.

2. Open `styles.css` and add some CSS to define the width and height of the slideshow:

   ```
   #slideshow  { width:600px;height:400px; }
   ```

3. Next, head over to `http://tobia.github.com/CrossSlide/` to get the downloads and documentation for the CrossSlide plugin.



CrossSlide

tobia.github.com/CrossSlide/

CrossSlide

A jQuery plugin to create pan and cross-fade animations

Project page – **Download minified** – Download source – Bugs, feature requests and support

CrossSlide is a jQuery plugin implementing in 2kB[1] of Javascript code some common slide-show animations, traditionally only available via Adobe Flash™ or other proprietary plugins. CrossSlide builds upon jQuery's animation facility, so it is as portable across browsers as jQuery itself (that is, a lot!)

Internally CrossSlide does not rely on lookup tables, building instead a "chain" of functions. Each function starts a linear phase of the animation through jQuery, setting the next function in the chain as a complete callback. After all the functions are prepared and the animation is started, no other calculations are made. This setup reduces runtime overhead to the absolute minimum, making CrossSlide the most optimized implementation of these effects.

CrossSlide can create a few different effects, depending on how it's called.

Static cross-fade

You'll find the **Download minified** link near the top of the page. The rest of the page shows several examples of the CrossSlide plugin in action. Take a look through the examples. You'll see that it can do everything from a simple crossfade slideshow similar to what we built in the first section of this chapter to a fully animated panning and zooming slideshow.

Now that you've had a look at some of the types of slideshows you can create with the CrossSlide plugin, here are a few things to keep in mind:

- First, because of the rendering limitations of some browsers (namely, Internet Explorer), zooming in and out on photos can affect the quality of the photo display. The plugin's author recommends keeping the zoom factor at or below 1 to minimize this effect.

- Second, because browsers are limited to rendering full pixels, the panning and zooming animation effects might be a bit less smooth than you'd like, particularly for diagonal animations. You can minimize the 1-pixel jumping effect by minimizing or avoiding diagonal animation or by choosing a relatively high speed for the animations, which helps them appear smoother.
- Finally, the animations can be a bit CPU-intensive, particularly when using the panning, zooming, and crossfading animations simultaneously, as we'll do in this example. It's nothing that should trip up most newer computers, but depending on your site's audience, you might want to avoid using all possible animation effects at once. At the end of this tutorial, I'll show you how to avoid the most CPU-intensive part of the slideshow if it's causing problems on your own or your site visitor's computers.

4. When you click the **Download minified** link, the plugin script itself will open in a browser window, just as jQuery itself does. Just right-click on the page or select **File | Save Page As** from the browser's menu bar to save the file to your own computer. Keep the file name, `jquery.cross-slide.min.js` and save the file in your `scripts` folder.

5. Next, we just have to include the CrossSlide plugin file at the bottom of our HTML page, between jQuery and `scripts.js`:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.cross-slide.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

6. Next, open your `scripts.js` file and we'll get started with the CrossSlide plugin by selecting our slideshow container and calling the `crossSlide()` method:

```
  var slideshow = $('#slideshow');
  slideshow.crossSlide();
```

Recall that a variable is just a container for something. In this case, we've selected the slideshow container and placed it in a variable called `slideshow`. We've done this because we're going to reference the container several times in our script. By saving the slideshow container in a variable, we're preventing jQuery from having to query the DOM looking for the slideshow container each time we want to refer to it, making our code more efficient.

7. At this point, if you load the page in a browser, you'll see that calling the `crossSlide()` method appears to have had no effect on our page. You'll still see the placeholder content inside our slideshow container and there's no slideshow happening. That's because we have to pass not only settings to the `crossSlide()` method, but also the list of photos we'd like to show in our slideshow. Inside the `crossSlide()` method's parentheses, insert a pair of curly brackets and we'll pass in a key/value pair to configure the length of time the fade between photos will take in seconds:

```
  var slideshow = $('#slideshow');
  slideshow.crossSlide({
    fade: 1
  });
```

Note that we're expressing the length of time in seconds, not milliseconds. The CrossSlide plugin is set up to expect seconds as units of time rather than the milliseconds that we usually find in JavaScript.

8. Next, after our configuration settings, we want to pass an array of photos to the `crossSlide()` method. An array is wrapped in square brackets:

```
slideshow.crossSlide({
  fade: 1
  }, [
    //Our list of photos will go here.
  ]
);
```

9. Each photo will have its own set of key/value pairs describing the URL of the image, the caption, and so on. Each photo will be wrapped in its own set of curly brackets. We'll get started with the URL of the photo which is described in the `src` key:

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src: 'images/1000/AguaAzul.jpg'
  }
  ]
);
```

10. Next, we'll add the caption for the photo as another key/value pair:

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src: 'images/1000/AguaAzul.jpg',
    alt: 'Agua Azul, Mexico'
  }
  ]
);
```

11. Now, we have to add two key/value pairs to describe the starting and ending points of the panning and zooming animation. Let's say we want to pan across this photo from the top left to the bottom right while zooming in. Here are the values we'll pass the `from` and `to` keys:

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src:  'images/1000/AguaAzul.jpg',
    alt:  'Agua Azul, Mexico',
    from:  'top left 1x',
    to:    'bottom right .8x'
  }
  ]
);
```

12. Finally, we want to specify how long the animation should take in seconds. I'll show this photo animation for four seconds:

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src:  'images/1000/AguaAzul.jpg',
    alt:  'Agua Azul, Mexico',
    from:  'top left 1x',
    to:    'bottom right .8x',
    time:  4
  }
  ]
);
```

13. That's one photo for our slideshow. To add more, simply add another set of key/value pairs inside curly brackets. Don't forget to separate each photo from the previous photo with a comma. Remember not to put a comma after the last photo in the list. Here's my example with three more photos added:

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src:  'images/1000/AguaAzul.jpg',
    alt:  'Agua Azul, Mexico',
    from:  'top left 1x',
    to:    'bottom right .8x',
    time:  4
  },
  {
    src: 'images/1000/BurneyFalls.jpg',
    alt: 'Burney Falls, California, USA',
    from:  'top left 1.2x',
    to:    'bottom right .8x',
    time:  5
  },
  {
    src: 'images/1000/Cachoeira_do_Pacheco.jpg',
    alt: 'Cachoeira do Pacheco, Venezuela',
    from:  '50% 0% 1.2x',
    to:    '50% 60% .6x',
    time:  4
  },
  {
    src: 'images/1000/Deer_Leap_Falls.jpg',
    alt: 'Deer Leep Falls, Pennsylvania, USA',
    from:  '50% 50% 1.2x',
    to:    '50% 100% .8x',
    time:  3
  }
  ]
);
```

Note that I can choose how long each photo displays—allowing a particularly stunning photo to linger on the page longer if I choose, or moving a smaller or less interesting photo off the page more quickly.

Now if you refresh the page in the browser, you'll see a panning and zooming slideshow of your photos. We're getting closer!

14. Next, we'll use that caption value we passed into the `crossSlide()` method for each photo to create a caption. First, I'm going to go back to my HTML markup and add a container for the caption. You can style this with CSS however you'd like:

```
<div id="slideshow">
  <img src="images/600/AguaAzul.jpg" alt="Agua Azul"/>
</div>
<div class="caption"></div>
```

Keep in mind that the container for your caption has to appear outside of the slideshow container. If you place it inside, it will be removed when the CrossSlide plugin replaces the slideshow container's content with the slideshow.

Now, we've got a place to display our caption, so we just need a way to put our captions into that container. The `crossSlide()` method will accept a callback method along with our settings and array of images. This callback function will be called each time an image starts to crossfade into the next image, and it is called again when the fade is complete.

```
slideshow.crossSlide({
  fade: 1
  }, [
  {
    src:  'images/1000/AguaAzul.jpg',
    alt:  'Agua Azul, Mexico',
    from:  'top left 1x',
    to:    'bottom right .8x',
    time:  4
  },
  {
    src: 'images/1000/BurneyFalls.jpg',
    alt: 'Burney Falls, California, USA',
    from:  'top left 1.2x',
    to:    'bottom right .8x',
    time:  4
  },
  {
    src: 'images/1000/Cachoeira_do_Pacheco.jpg',
    alt: 'Cachoeira do Pacheco, Venezuela',
    from:  '50% 0% 1.2x',
    to:    '50% 60% .6x',
    time:  4
  },
  {
    src: 'images/1000/Deer_Leap_Falls.jpg',
    alt: 'Deer Leep Falls, Pennsylvania, USA',
    from:  '50% 50% 1.2x',
    to:    '50% 100% .8x',
    time:  3
  }
  ], function(index, img, indexOut, imgOut) {
    //our callback function goes here
  }
);
```

Our callback function is passed four possible values: the index of the current image, the current image itself, the index of the previous image, and the previous image itself. The index of the image is simply its place in the slideshow by number. JavaScript, like other programming languages, starts counting at 0 instead of 1. So the index of the first image in the slideshow is 0, the second image's index is 1, and so on.

Remember I said the callback function is called once when the crossfade starts and once again after the crossfade is finished? If the crossfade is starting, the callback function will get all four values—the index of and the current image, and the index of and the previous image. If the crossfade is finished, we'll only get two values: the index of the current image and the current image itself.

15. We'll check to see if the crossfade is starting or finishing. If the crossfade is finished, then we'll want to show the caption for the new photo. If the crossfade is just starting, then we'll hide the caption for what will very soon be the previous image:

```
    ], function(index, img, indexOut, imgOut) {
      var caption = $('div.caption');
      if (indexOut == undefined) {
        caption.text(img.alt).fadeIn();
      } else {
        caption.fadeOut();
      }
    }
```

If the crossfade is finished, then `indexOut` will be `undefined`, since there won't be a value for that variable passed to the callback function. It's easy to check if that value is `undefined` to figure out if the crossfade animation is starting or finishing. Then, we use jQuery's `text()` method to set the text of the caption to the `alt` value we included with each image and fade the caption in. If the crossfade animation is just starting on the other hand, we'll just fade the caption out.

Now if you refresh the page in the browser, you'll see that the caption fades in with each photo and fades out as the crossfade is starting. It's a nice smooth transition from one caption to the next.

16. This last step is optional. If you find that the CrossSlide plugin with all animations running at once, as we've set up in this example, is too CPU-intensive for your computer or the computers of your site visitors, there's a simple configuration option that will allow you to skip the most CPU-intensive part of the slideshow—namely, when two photos are crossfading while panning and zooming. All you have to do is pass another key/value pair to the configuration options setting `variant` to `true`:

    ```
    slideshow.crossSlide({
      fade: 1,
      variant: true
      }, [
      {
        src:  'images/1000/AguaAzul.jpg',
    ...
    ```

    This will change your slideshow so that each photo will complete panning and zooming before starting the crossfade to the next photo.

What just happened?

Don't be worried if your head is spinning—the CrossSlide plugin is by far the most developer-y plugin we've used yet. Although this plugin isn't super designer-friendly, I hope you can see that even this type of plugin is within your reach if you have a little patience and are willing to experiment a bit. Carefully studying the code for examples will take you pretty far.

We set up a container which held our static content for users with JavaScript disabled. Then we set up the CrossSlide plugin to replace that content with a dynamic panning and zooming slideshow for the users with JavaScript enabled. We set the length of the crossfade to 1 second, and then passed in our array of images, including the URL, caption, animation starting point, animation ending point, and duration for each image. Finally, we took advantage of the callback function provided by the CrossSlide plugin to fade in each photo's caption and fade it back out when the photo itself starts to fade out. We also took a look at how to make the slideshow a bit less CPU-intensive for those situations where it might cause problems.

Summary

We took a look at four different approaches to building photo slideshows with jQuery. We started off with a simple crossfading slideshow that we built from scratch without a plugin. We explored fancy transition effects with the Nivo Slider plugin. Then we learned how to set up a thumbnail slideshow using the Galleriffic plugin. And finally, we took a look at building a panning and zooming slideshow with the CrossSlide plugin.

Next up, we'll take a look at building sliders and carousels for all types of content on your site.

10
Featuring Content in Carousels and Sliders

In addition to slideshows, we can also feature images and text in sliders and carousels. One or more slides can be visible at one time and a sliding animation is used for transition between the slides. Carousels are ideal for creating a featured content slider or for making many images available in a small space. We'll take a look at the flexible and customizable jCarousel plugin from Jan Sorgalla and how it can be used to create several different types of carousel and slider solutions.

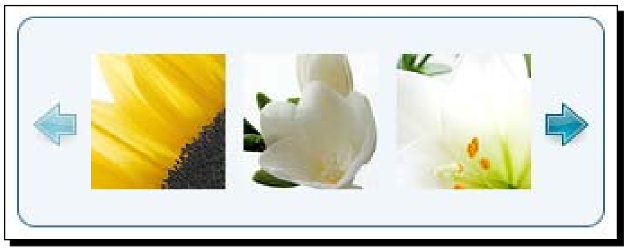
In this chapter, we'll learn the following topics:

- Using the jCarousel plugin to create a basic horizontal slider
- Creating a vertical news ticker
- Creating a featured content slider with external controls
- Combining a slideshow with a thumbnail carousel

Basic jCarousel

Let's first take a look at creating a basic horizontal carousel of image thumbnails. The jCarousel plugin includes two different skins, so setting up a basic carousel is quick and easy.

The following screenshot is a sample of a basic carousel using the tango skin that's included with the plugin:



There are a dozen or so thumbnail images in the carousel. Clicking one of the side arrows slides the carousel left or right to reveal the next set.

Time for action – creating a basic carousel

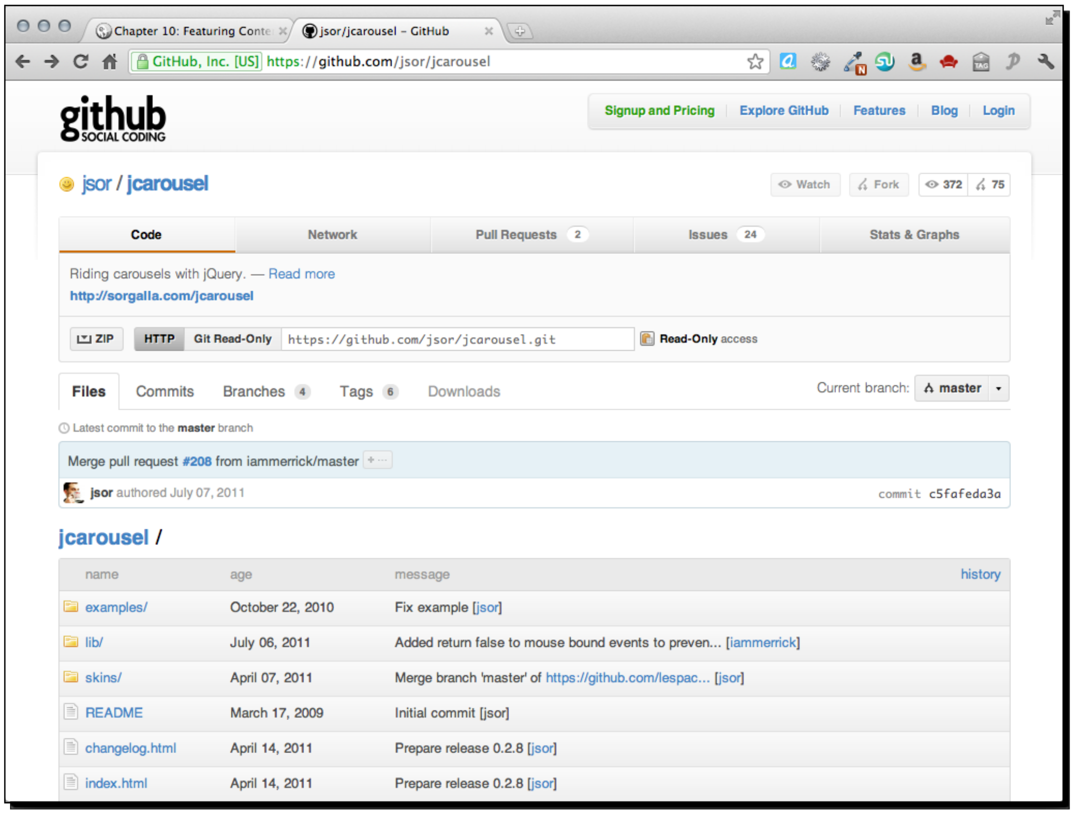
Follow these steps to set up a basic jCarousel of images:

1. As usual, we'll get started with our HTML. Set up a basic HTML document and associated files and folders just like we did in *Chapter 1*, *Designer, Meet jQuery*. In the body of the HTML document, create an unordered list of images. The carousel works best when the images are of uniform size. I've made my images 200 pixels wide by 150 pixels tall. Here's what my HTML looks like:

```
<ul id="thumb-carousel">
  <li><img src="images/thumbs/Switzerland.png"
alt="Switzerland"/></li>
  <li><img src="images/thumbs/CostaRica.png" alt="Costa Rica"/></
li>
  <li><img src="images/thumbs/Canada.png" alt="Canada"/></li>
  <li><img src="images/thumbs/Seychelles.png" alt="Seychelles"/></
li>
  <li><img src="images/thumbs/Tuvalu.png" alt="Tuvalu"/></li>
  <li><img src="images/thumbs/Iceland.png" alt="Iceland"/></li>
  <li><img src="images/thumbs/SouthAfrica.png" alt="South
Africa"/></li>
  <li><img src="images/thumbs/Mexico.png" alt="Mexico"/></li>
  <li><img src="images/thumbs/Spain.png" alt="Spain"/></li>
  <li><img src="images/thumbs/Italy.png" alt="Italy"/></li>
  <li><img src="images/thumbs/Australia.png" alt="Australia"/></
li>
  <li><img src="images/thumbs/Argentina.png" alt="Argentina"/></
li>
</ul>
```

You can see that I've assigned an `id` of `thumb-carousel` to my unordered list, and that the HTML is simple and straightforward: just a list of images.

2. Next, we'll need to download the jCarousel plugin. The plugin is available for download from GitHub here: `https://github.com/jsor/jcarousel`.



Riding carousels with jQuery. — Read more
http://sorgalla.com/jcarousel

Latest commit to the **master** branch

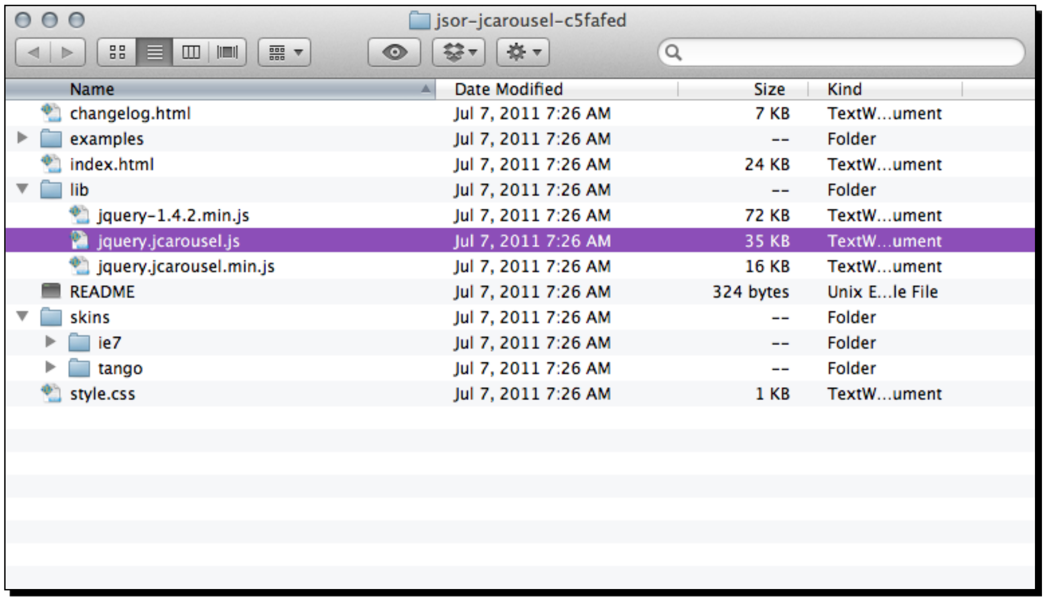
Merge pull request #208 from iammerrick/master
jsor authored July 07, 2011 commit c5fafeda3a

jcarousel /

name	age	message
examples/	October 22, 2010	Fix example [jsor]
lib/	July 06, 2011	Added return false to mouse bound events to preven... [iammerrick]
skins/	April 07, 2011	Merge branch 'master' of https://github.com/lespac... [jsor]
README	March 17, 2009	Initial commit [jsor]
changelog.html	April 14, 2011	Prepare release 0.2.8 [jsor]
index.html	April 14, 2011	Prepare release 0.2.8 [jsor]

To download the plugin, just click on the **ZIP** button.

3. Next, unzip the folder and have a look inside.



Name	Date Modified	Size	Kind
changelog.html	Jul 7, 2011 7:26 AM	7 KB	TextW...ument
examples	Jul 7, 2011 7:26 AM	--	Folder
index.html	Jul 7, 2011 7:26 AM	24 KB	TextW...ument
lib	Jul 7, 2011 7:26 AM	--	Folder
jquery-1.4.2.min.js	Jul 7, 2011 7:26 AM	72 KB	TextW...ument
jquery.jcarousel.js	Jul 7, 2011 7:26 AM	35 KB	TextW...ument
jquery.jcarousel.min.js	Jul 7, 2011 7:26 AM	16 KB	TextW...ument
README	Jul 7, 2011 7:26 AM	324 bytes	Unix E...le File
skins	Jul 7, 2011 7:26 AM	--	Folder
ie7	Jul 7, 2011 7:26 AM	--	Folder
tango	Jul 7, 2011 7:26 AM	--	Folder
style.css	Jul 7, 2011 7:26 AM	1 KB	TextW...ument

Inside, we'll find a folder called `examples`, which contains many examples of the jCarousel plugin in action. There's an `index.html` file that contains the documentation for the plugin. A `skins` folder contains the two skins that are included with the plugin along with the images that those skins require. And finally, a `lib` folder contains jQuery, and two copies of the jCarousel plugin—one minified and one not.

4. We're going to use the `tango` skin and the minified version of the plugin. Copy `jquery.jcarousel.min.js` to your own `scripts` folder and copy the entire `tango` folder to your own `styles` folder.

5. Next, we need to attach the CSS and JavaScript to our HTML file. In the `<head>` section of the document, attach the tango skin's CSS file before your own `styles.css` file:

```
<link rel="stylesheet" href="styles/tango/skin.css"/>
<link rel="stylesheet" href="styles/styles.css"/>
```

6. At the bottom of the document, just before the closing `</body>` tag, attach the jCarousel plugin file after jQuery and before your own `scripts.js`:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.jcarousel.min.js"></script>
<script src="scripts/scripts.js"></script>
```

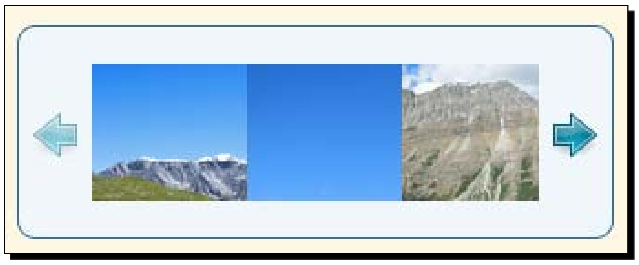
7. The tango skin for the jCarousel slider is dependent on the `jcarousel-skin-tango` class placed on a wrapper for the list. Wrap the list in a `div` tag and give the `div` the appropriate class:

```
<div class="jcarousel-skin-tango">
  <ul id="thumb-carousel">
  ...
  </ul>
</div>
```

8. The next thing we'll do is set up our own JavaScript. Open your `scripts.js` file. Call the `ready` method on the document, select the image list, and call the `jcarousel()` method:

```
  $(document).ready(function(){
    $('#thumb-carousel').jcarousel();
  });
```

As usual, calling the `jcarousel()` method this way will load the carousel with all of the default settings. Refresh the page in the browser and this is what you'll see:



Not exactly what we had in mind, but clicking the next arrow on the right will advance the carousel. Let's take a look at getting some custom settings in place so that we can view our complete images as intended.

9. The CSS for the tango skin assumes that our images are 75 pixels wide by 75 pixels tall, but that's not the case with our carousel. We'll add a few lines of CSS to our `styles.css` to adjust the size of our images. First, we'll specify the width and height of a single item:

```
.jcarousel-skin-tango .jcarousel-item { width:200px;height:150px;}
```

10. We'll also have to adjust the overall size of the carousel's container and clip container:

```
.jcarousel-skin-tango .jcarousel-clip-horizontal {
width:830px;height:150px;}
.jcarousel-skin-tango .jcarousel-container-horizontal {
width:830px; }
```

You might wonder where that 830px-wide measurement came from. Each item is 200 pixels wide, and there are 10 pixels between each image.

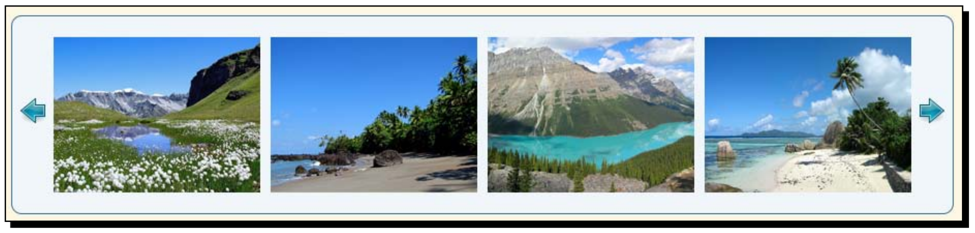
200 + 10 + 200 + 10 + 200 + 10 + 200 = 830

The total width of images and the gaps between them is 830 pixels.

11. Next, we'll need to bump the next and previous buttons down a bit since our carousel is taller than the default and the buttons appear too high:

```
.jcarousel-skin-tango .jcarousel-prev-horizontal,
.jcarousel-skin-tango .jcarousel-next-horizontal { top:75px; }
```

Now the carousel looks just the way we'd like:



12. Finally, we'll make a few adjustments to the settings for the jCarousel plugin itself. Like many other plugins, we can make customizations by passing a set of key/value pairs to the `jcarousel()` method inside a pair of curly braces. First, let's change the `scroll` value to `4` so that four items will scroll each time we press the next or previous button. Go back to your `scripts.js` file and add the new key/value pair to your script as follows:

```
$('#thumb-carousel').jcarousel({
  scroll: 4
});
```

Next, the carousel currently reaches a hard stop at either the beginning or the end. Instead, we'll make the carousel wrap—if the site visitor is viewing the last item in the carousel and presses the next button, the carousel will wrap back to the beginning. And vice versa if the back button is clicked while viewing the first item. We'll add a `'both'` value for the `wrap` key so that the carousel will wrap at either end:

```
$('#thumb-carousel').jcarousel({
  scroll: 4,
  wrap: 'both'
});
```

Refresh the page in the browser and page through the carousel using either the next or the previous button or any combination of the two. That's all there is to creating a simple carousel with the jCarousel plugin.

What just happened?

We used the jCarousel plugin to create a basic animated image thumbnail carousel. We used one of the default skins included with the plugin and made adjustments for the size of our content with CSS. A few simple customizations were passed to the carousel to make sure it worked just the way we wanted.

Animated news ticker

A horizontal image carousel is nice, but it has pretty limited use. Luckily, the jCarousel plugin is flexible enough to be used for a variety of different purposes. In this section, we'll learn how to create an animated news ticker.

Time for action – creating an animated news ticker

Follow these steps to set up a vertical news listing:

1. First, up, we'll set up a basic HTML file and associated files and folders like we did in *Chapter 1, Designer, Meet jQuery*. In the body of the HTML document, create an unordered list of news items. Each news item will have an image and a div that contains a headline and an excerpt:

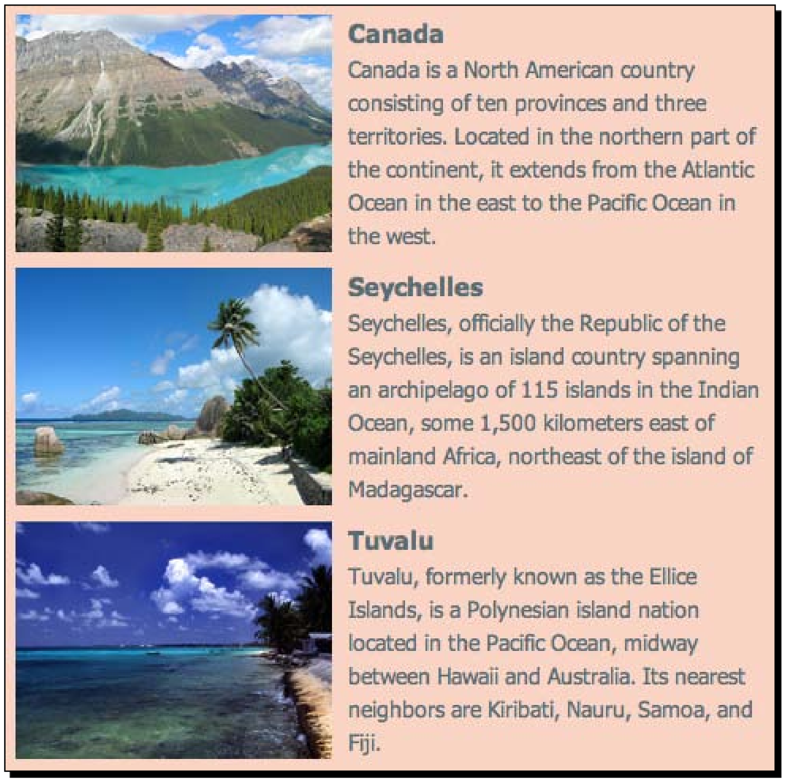
```
<ul id="news-carousel">
  <li>
    <img src="images/thumbs/Switzerland.png" alt="Switzerland"/>
    <div class="info">
      <h4>Switzerland</h4>
      <p>Switzerland, officially the Swiss Confederation, is a
federal republic consisting of 26 cantons, with Bern as the seat
of the federal authorities</p>
    </div>
  </li>
  <li>
    <img src="images/thumbs/CostaRica.png" alt="Costa Rica"/>
    <div class="info">
      <h4>Costa Rica</h4>
      <p>Costa Rica, officially the Republic of Costa Rica, is a
country in Central America, bordered by Nicaragua to the north,
Panama to the south, the Pacific Ocean to the west and south and
the Caribbean Sea to the east.</p>
    </div>
  </li>
  ...
</ul>
```

I've created 12 items in total on my list, each with this same structure. Keep in mind that each item in the carousel must be of the same width and height.

2. Next up, we'll open our `styles.css` file and add a few lines of CSS to get each news item styled the way we'd like, with the image on the left and the headline and excerpt on the right:

```
#news-carousel li  { overflow:hidden;zoom:1;list-style-type:none;
}
#news-carousel li img  { float:left; }
#news-carousel li .info  { margin-left:210px; }
#news-carousel h4  { margin:0;padding:0; }
#news-carousel p  { margin:0;padding:0;font-size:14px; }
```

Feel free to add some additional CSS to style the list to suit your own taste. If you open the page in a browser, at this point, you can expect to see something similar to the following screenshot:



Canada

Canada is a North American country consisting of ten provinces and three territories. Located in the northern part of the continent, it extends from the Atlantic Ocean in the east to the Pacific Ocean in the west.

Seychelles

Seychelles, officially the Republic of the Seychelles, is an island country spanning an archipelago of 115 islands in the Indian Ocean, some 1,500 kilometers east of mainland Africa, northeast of the island of Madagascar.

Tuvalu

Tuvalu, formerly known as the Ellice Islands, is a Polynesian island nation located in the Pacific Ocean, midway between Hawaii and Australia. Its nearest neighbors are Kiribati, Nauru, Samoa, and Fiji.

3. Just as in our simple carousel example, we'll attach the tango skin CSS in the `<head>` section of the document, and the jCarousel plugin script at the bottom of the document, between jQuery and our own `scripts.js` file.

4. Next, open your `scripts.js` file. We'll write our document ready statement, select our news ticker, and call the `jcarousel()` method, just like we did in the previous example.

```
$(document).ready(function(){
  $('#news-carousel').jcarousel();
});
```

5. We'll pass some customization options to the `jcarousel()` method to adjust our carousel to work the way that we'd like. First, it should be vertical rather than horizontal, so pass `true` as a value for the `vertical` key:

   ```
   $('#news-carousel').jcarousel({
     vertical:true
   });
   ```

6. We'd also like to scroll only one item at a time:

   ```
   $('#news-carousel').jcarousel({
     vertical:true,
     scroll:1
   });
   ```

7. And, we'd like the list of news items to loop endlessly as follows:

   ```
   $('#news-carousel').jcarousel({
     vertical:true,
     scroll:1,
     wrap:'circular'
   });
   ```

8. We'd like the carousel to automatically advance through the news stories in true news-ticker fashion. We'll advance the carousel every three seconds:

   ```
   $('#news-carousel').jcarousel({
     vertical:true,
     scroll:1,
     wrap:'circular',
     auto: 3

   });
   ```

9. And last but not least, we'll slow the animation down a bit so that it's less jarring in case our site visitor is in the middle of reading when the animation is triggered. 600 milliseconds ought to be slow enough:

   ```
   $('#news-carousel').jcarousel({
     vertical:true,
     scroll:1,
     wrap:'circular',
     auto: 3,
     animation: 600
   });
   ```

10. Now that we've got jCarousel configured just the way we'd like, all that's left to do is customize the appearance of the carousel. We're currently using the default tango skin, which is still assuming our individual items are 75 pixels wide by 75 pixels tall. Open your `styles.css` file and we'll get started by adjusting the necessary widths and heights as follows:

```
.jcarousel-skin-tango .jcarousel-item  { width:475px;height:150px;
}
.jcarousel-skin-tango .jcarousel-clip-vertical  {
width:475px;height:470px; }
.jcarousel-skin-tango .jcarousel-container-vertical  {
height:470px;width:475px; }
```

We've set the size of an individual item to 475 pixels wide by 150 pixels tall. Then the size of the container and clip container are adjusted to show three items. Just as a reminder—since each item in our carousel is 150 pixels tall and there are 10 pixels of space between items, we can calculate the height of the container as follows:

150 + 10 + 150 + 10 + 150 = 470px

We're using heights instead of widths for our calculations since our carousel is now vertical rather than horizontal.

11. Next, we'll adjust the tango style a bit to fit in with my site's design. I'm going to start by replacing the pale blue color scheme of the container with an orange color scheme, and adjust the rounded corners to be a bit less round:

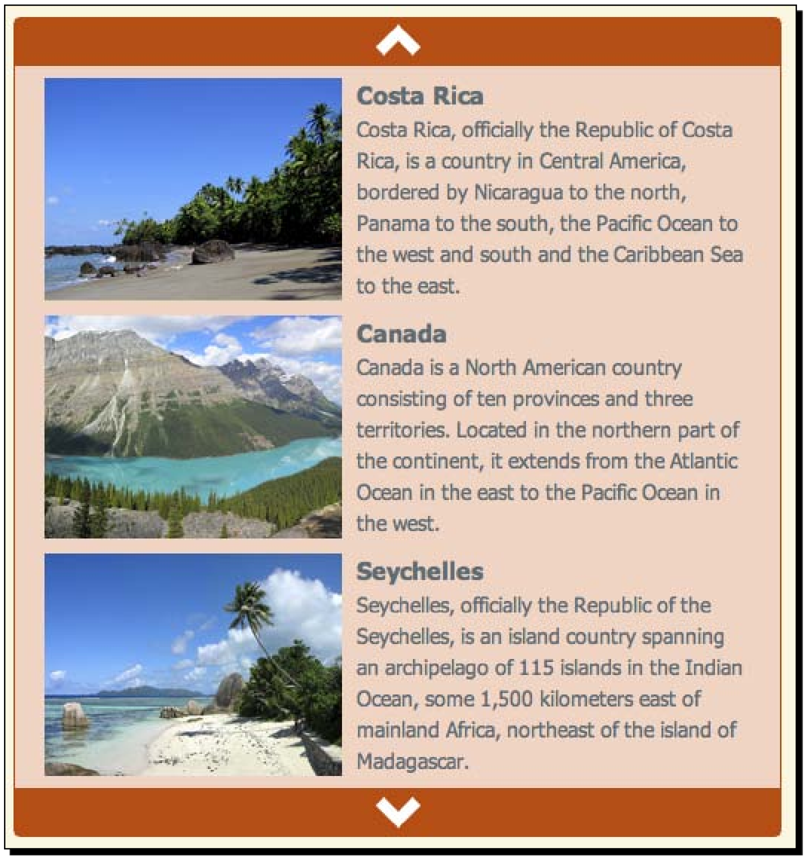
```
.jcarousel-skin-tango .jcarousel-container  { -moz-border-radius:
5px;-webkit-border-radius:5px;border-radius:5px;border-color:#CB4B
16;background:#f9d4c5; }
```

12. Now, let's replace the small blue arrows of the tango skin with a long orange bar that spans the full width of our carousel. I've created my own arrow graphic that I'll show in the middle of each button:

```
.jcarousel-skin-tango .jcarousel-prev-vertical,
.jcarousel-skin-tango .jcarousel-next-vertical  {
left:0;right:0;width:auto; }
.jcarousel-skin-tango .jcarousel-prev-vertical  {
top:0;background:#cb4b16 url(images/arrows.png) 50% 0 no-repeat; }
.jcarousel-skin-tango .jcarousel-prev-vertical:hover,
.jcarousel-skin-tango .jcarousel-prev-vertical:focus  {
background-color:#e6581d;background-position:50% 0; }
.jcarousel-skin-tango .jcarousel-next-vertical  {
background:#cb4b16 url(images/arrows.png) 50% -32px no-
repeat;bottom:0; }
```

```
.jcarousel-skin-tango .jcarousel-next-vertical:hover,
.jcarousel-skin-tango .jcarousel-next-vertical:focus { background-
color:#e6581d;background-position:50% -32px; }
```

Now, if you refresh the page in the browser, you'll see that the carousel is re-designed a bit with a different color scheme and appearance:



Costa Rica
Costa Rica, officially the Republic of Costa Rica, is a country in Central America, bordered by Nicaragua to the north, Panama to the south, the Pacific Ocean to the west and south and the Caribbean Sea to the east.

Canada
Canada is a North American country consisting of ten provinces and three territories. Located in the northern part of the continent, it extends from the Atlantic Ocean in the east to the Pacific Ocean in the west.

Seychelles
Seychelles, officially the Republic of the Seychelles, is an island country spanning an archipelago of 115 islands in the Indian Ocean, some 1,500 kilometers east of mainland Africa, northeast of the island of Madagascar.

Moving your mouse over the top or bottom bar will lighten the color a bit, and clicking a bar will advance the carousel in that direction by one item.

What just happened?

In this case, we used the jCarousel plugin to create a vertical news ticker. Our news ticker automatically advances one item every three seconds. We slowed down the animation to make for a smoother reading experience for our site visitors. We also saw how we can customize the tango skin's CSS to customize the color scheme and appearance of the carousel to fit our site's design. Next up, we'll take a look at how we can add some external controls to the carousel.

Have a go hero – design your own carousel

Now that you've seen how to customize the appearance and behavior of the jCarousel plugin, design your own carousel. It could be horizontal or vertical, contain text, images, or a combination of both. Experiment with the settings that the jCarousel plugin makes available to you—you'll find them all listed out and explained in the plugin's documentation.

Featured content slider

In addition to carousels that show multiple items at one time, jCarousel can also be used to build content sliders that show just one item at a time. It's also possible to build external controls that add some additional functionality to your carousels. Let's take a look at how to create a single-slide featured content slider with external pagination controls.

Time for action – creating a featured content slider

We'll get started as usual by setting up our basic HTML file and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*.

1. In the body of the HTML document, the HTML markup for our featured content slider will be very similar to the HTML we set up for a news ticker. The only difference is that I'm replacing the images with larger images since I want images to be the main focus of the slider. I'm using images that are 600 pixels wide by 400 pixels tall. The following is a sample of the HTML:

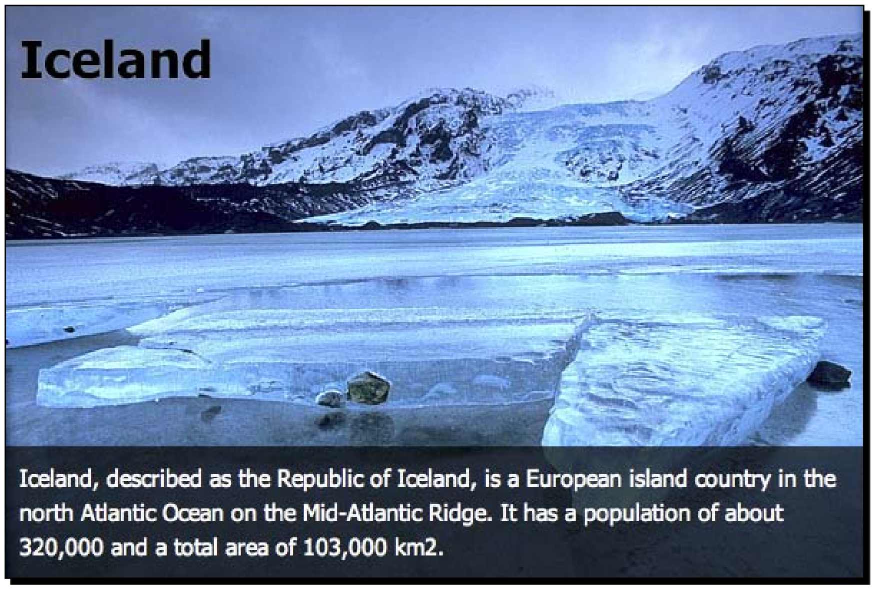
```
<div class="jcarousel-skin-slider">
  <ul id="featured-carousel">
    <li>
      <a href="#"><img src="images/600/Switzerland.jpg" 
alt="Switzerland"/></a>
      <div class="info">
        <h4>Switzerland</h4>
        <p>Switzerland, officially the Swiss Confederation, is a
federal republic consisting of 26 cantons, with Bern as the seat
of the federal authorities</p>
      </div>
    </li>
    <li>
      <a href="#"><img src="images/600/CostaRica.jpg" alt="Costa
Rica"/></a>
      <div class="info">
        <h4>Costa Rica</h4>
        <p>Costa Rica, officially the Republic of Costa Rica, is
a country in Central America, bordered by Nicaragua to the north,
Panama to the south, the Pacific Ocean to the west and south and
the Caribbean Sea to the east.</p>
      </div>
    </li>
    ...
  </ul>
</div>
```

I have 12 items in total on my list, each marked up just the way you see here. Note that I've wrapped my list in a `div` with the class `jcarousel-skin-slider`. We'll be using this class to style our list with CSS.

2. Next up, we'll style our list of items. We'll overlay the headline and paragraph of text on the photo, the header along the top, and the paragraph of text along the bottom. The following is the CSS we can use to accomplish that:

```
#featured-carousel li  { overflow:hidden;list-style-type:none;posi
tion:relative;width:600px;height:400px; }
#featured-carousel h4  { position:absolute;top:0;left:0;right:0;pa
dding:10px;margin:0;color:#000;font-size:36px;text-shadow:#fff 0 0
1px; }
#featured-carousel p  { position:absolute;bottom:0;left:0;right:
0;padding:10px;margin:0;color:#fff;background:#000;background:rg
ba(0,0,0,0.7); }
```

Now each item in my list looks similar to the following screenshot:



Iceland, described as the Republic of Iceland, is a European island country in the north Atlantic Ocean on the Mid-Atlantic Ridge. It has a population of about 320,000 and a total area of 103,000 km2.

I want to draw your attention to a couple of handy CSS tricks I've put to use here. First, notice that I've added a small white `text-shadow` to the headline and have made the headline text black. Just in case this text happens to overlay a dark area of the image, the subtle white outline around the text will help the text to stand out. Then, note that I've added two background values for the short paragraph of text. The first, a solid black, the second a transparent black color denoted with an `rgba` value. The first value is for versions of Internet Explorer before IE9. Those browsers will display a solid black background. Newer and more capable browsers will use the second value—the `rgba` value—to display a slightly transparent black background behind the text—allowing the image to show through a bit while making the text more readable.

3. Now, we'll attach the jCarousel JavaScript at the bottom of the page, between jQuery and our `scripts.js` file, just as we've done in the other examples in this chapter.

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.jcarousel.min.js"></script>
<script src="scripts/scripts.js"></script>
```

4. Now we're going to write a bit of CSS to customize the appearance of our content slider. Open your `styles.css` file and add the following styles:

```
.jcarousel-skin-slider .jcarousel-container-horizontal { width:
600px; }
.jcarousel-skin-slider .jcarousel-clip { overflow: hidden; }
.jcarousel-skin-slider .jcarousel-clip-horizontal {
width:600px;height:425px; }
.jcarousel-skin-slider .jcarousel-item { width:600px;height:400px;
}
```

   Yep, that's really it. Just a few lines. We'll set the width of an individual item, the container, and the clip container to 600 pixels, the same as the width of one image. The height of the individual item is also set to 400 pixels, but we're going to set the clip container's height to 425 pixels to give us 25 pixels to add in some external controls, which we'll be looking at in a minute.

5. Now, open up your `scripts.js` file. The first thing we want to do is select our list and store it in a variable. This is because we're going to be using the list multiple times, and we don't want jQuery to have to query the DOM looking for our list each time.

```
  var slider = $('#featured-carousel');
```

6. Next, we'll set up our document ready statement and call the `jcarousel()` method on the slider, and we'll tell it that we want to scroll one pane at a time:

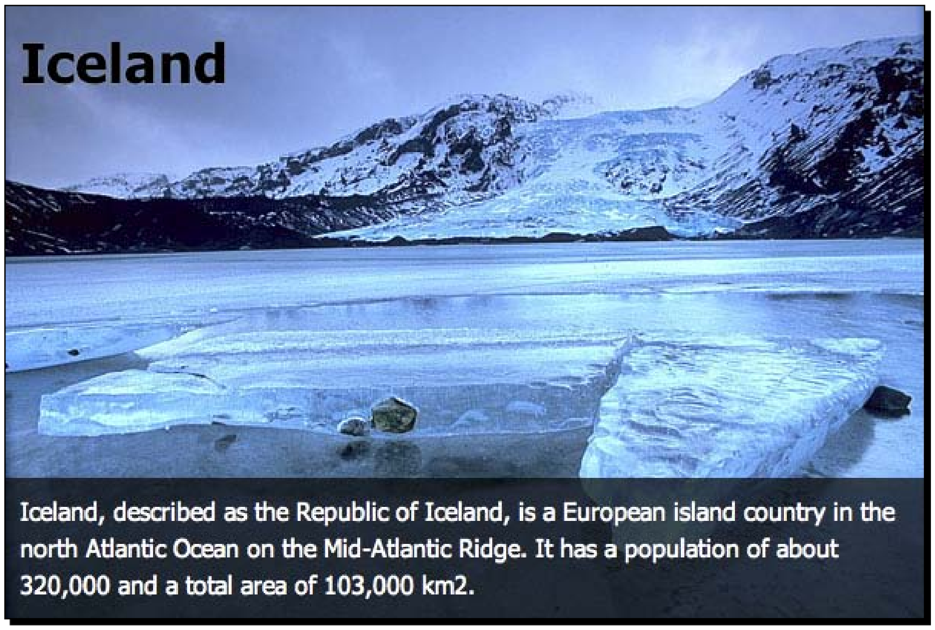
```
var slider = $('#featured-carousel');

$(document).ready(function(){
  slider.jcarousel({
    scroll: 1
  });
});
```

7. We're going to be adding our own external controls, so we'll need to remove the ones that the `jcarousel()` method creates on its own. Here's how we can do that:

```
$(document).ready(function(){
  slider.jcarousel({
    scroll: 1,
    buttonNextHTML: null,
    buttonPrevHTML: null
  });
});
```

The `buttonNextHTML` and `buttonPrevHTML` keys are provided so that you can specify your own HTML markup for those buttons. In this case, we're passing `null` as the value for both keys which will prevent them from being created.

Now we've done the basics to set up our slider. If you look at the page in your browser, you'll see the first slide. We haven't yet provided a way to navigate to the other slides, so let's jump on that next.



Iceland, described as the Republic of Iceland, is a European island country in the north Atlantic Ocean on the Mid-Atlantic Ridge. It has a population of about 320,000 and a total area of 103,000 km2.

Pagination controls

We've set up a basic slider that shows one item at a time, but you've no doubt noticed that there isn't a way to get to view any slide other than the first one. We removed jCarousel's default next and previous buttons, and we haven't provided any alternative yet. Let's add in some pagination controls so our site visitors can get to any slide they like.

Time for action – adding pagination controls

Next, we want to set up the function that will create the next button, previous button, and pagination buttons and makes them work.

1. The jCarousel plugin provides a key called `initCallback` that will allow us to pass in the name of a function that should be called when the carousel is created. Let's get started by creating an empty function and calling it:

   ```
   var slider = $('#featured-carousel');

   function carouselInit(carousel) {
     // Our function goes here
   }

   $(document).ready(function(){
     slider.jcarousel({
       scroll: 1,
       buttonNextHTML: null,
       buttonPrevHTML: null,
       initCallback: carouselInit
     });
   });
   ```

   Whatever actions we write inside of our `carouselInit()` function, it will be executed when the carousel is initialized or set up. Since any page numbers and previous and next buttons would only be functional if JavaScript is enabled, we want to create those buttons dynamically with JavaScript rather than coding them in our HTML. Let's take a look at how we can create a list of page links to each slide in the slider.

2. We'll get started by getting all of the slides in our slider. Remember that our slider is an unordered list and each slide in the slider is an individual list item in the list. Since we've already saved a reference to the slider itself, we can get all the slides inside of it as follows:

   ```
   function carouselInit(carousel) {
     var slides = slider.find('li');
   }
   ```

3. We'll use these slides in a moment to create the page numbers. In the meantime though, we need a place to put our page numbers, so let's create a couple of containers before the slider so that our pagination will display just above the slider. Here's how we insert two nested `<div>` tags just before the slider:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
}
```

4. Next, we'll need to refer to these two newly created containers a couple of times in our code, so we'll store references to them in variables as shown in the following code:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
}
```

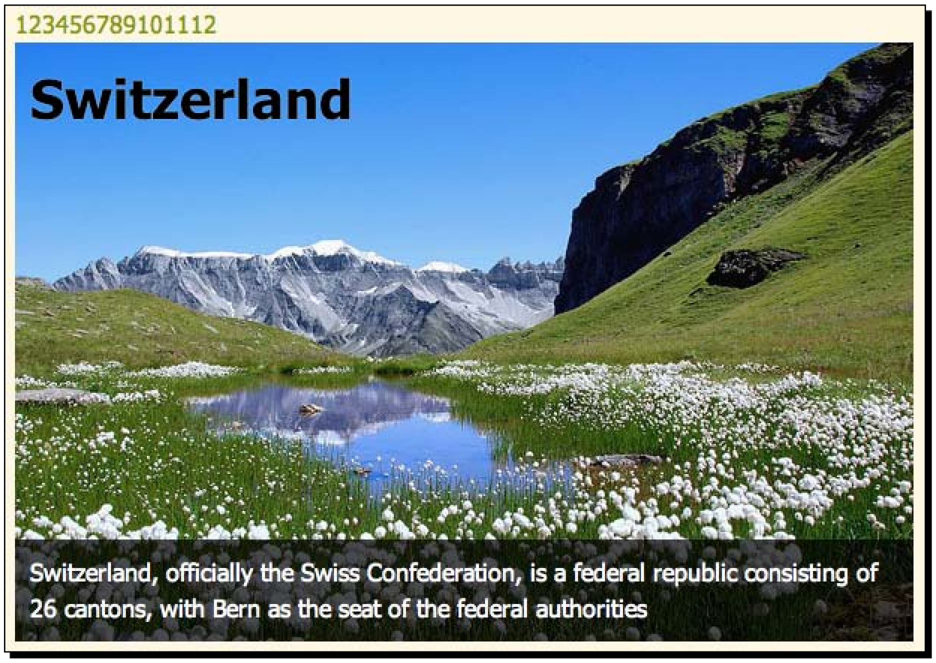
5. Now, we're going to get fancy and create a page number for each slide in the slider. The following is the code we'll add:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }
}
```

We're starting with `i = 1`, because the first page number will be 1. Then we're checking to see if `i` is less than or equal to the number of slides (`slides.length` is the number of slides). If `i` is less than or equal to the number of slides, we're going to increment `i` by one number—basically we're going to add 1 to `i` and `i++` is a JavaScript shortcut way of saying `i = i+1`.

Each time through the loop, we're going to append a link to the pages container we created. It's a link wrapped around a page number, and `i` represents our page number.

If you refresh the page in a browser at this point, you'll see numbers 1 to 12 linked above the slideshow. They aren't styled, and clicking on them won't do anything, because we haven't set that up yet—that's what we'll do next.

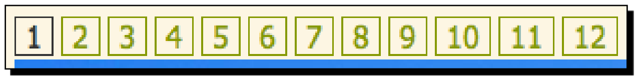


Switzerland, officially the Swiss Confederation, is a federal republic consisting of 26 cantons, with Bern as the seat of the federal authorities

6. Next, we want to style the links so that they look the way we'd like. Open up your `styles.css` file and add these few lines to the CSS:

```
#page-controls  { line-height:25px;height:25px; }
#page-controls a  { margin:0 4px 0 0;padding:0 5px;border:1px
solid #859900; }
#page-controls a:hover  { border-color: #D33682; }
#page-controls a.current  { color:#333;border-color:#333; }
```

This sets the height for our slider controls row to the 25 pixels that we allowed for it previously. Then we put a green border around each link, which will turn to a pink border when the link is hovered over. We adjusted margins and padding to get a nicely spaced row of boxes. Finally, we added a `.current` class for our links to allow us to mark the currently selected link in dark gray.



7. Okay, we have our page numbers added to our document, so all we have to do is make them work. We'll bind a click function to those links, since we want something to happen when our site visitor clicks on the links. We'll get started as follows:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }

  pages.find('a').bind('click', function(){
    //click code will go here
  });
}
```

8. The first thing to do inside our function is to cancel the default action of the click so that the browser doesn't try to do its own thing when the links are clicked.

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }

  pages.find('a').bind('click', function(){
    return false;
  });
}
```

9. The jCarousel plugin offers us a nice way to scroll to a particular slide in the slider. It looks as follows:

```
carousel.scroll($.jcarousel.intval(number));
```

The `number` near the end is where we would pass in which slide we want to scroll to. For example, if we wanted to scroll to the sixth slide, we'd say:

```
carousel.scroll($.jcarousel.intval(6));
```

In our case, the number slide we want to scroll to is the page number in our link. For example, if I click on the following link:

```
<a href="#">3</a>
```

That means I want to scroll to the third slide in the slider. I can get that number by using jQuery's `text()` method as follows:

```
pages.find('a').bind('click', function(){
  carousel.scroll($.jcarousel.intval($(this).text()));
  return false;
});
```

If I click on the fourth link, `$(this).text()` will be equal to 4; on the seventh link, it will be equal to 7, and so on.

Refresh the page in the browser, and you'll see that clicking on a numbered link will scroll the slider to that slide.

10. Clicking on the page numbers, you probably noticed that the current page number isn't highlighted in the pagination. We already wrote the CSS to highlight a link that has the `current` class – now we just have to be sure we're adding that class to the current link. Here's how we'll do that.

```
pages.find('a').bind('click', function(){
  carousel.scroll($.jcarousel.intval($(this).text()));
  $(this).addClass('current');
  return false;
});
```

Now if you refresh the page in the browser, you'll see that clicking a page number applies the `current` class CSS to the link, highlighting it. However, clicking a second page number highlights that link in addition to the previous link. We have to make sure that we're removing the class from the old link too. Add the following line to take care of that:

```
pages.find('a').bind('click', function(){
  carousel.scroll($.jcarousel.intval($(this).text()));
  $(this).siblings('.current').removeClass('current');
  $(this).addClass('current');
  return false;
});
```

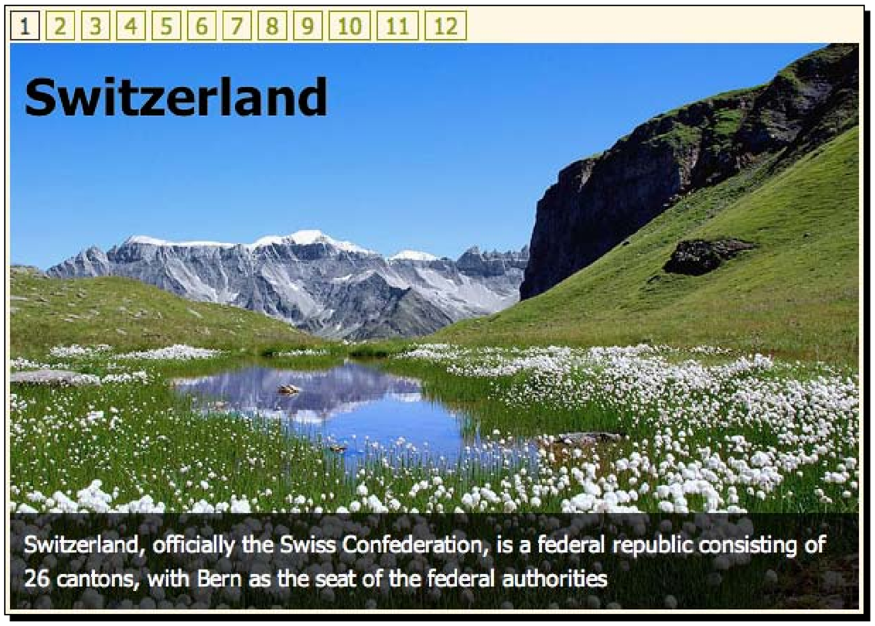
This line checks all of the links' siblings for any that might have the class of `current`. If it finds any, it removes the class.

11. Now, we just have to make sure the first link is highlighted when the carousel is initialized. The easiest way to do that is to simply click the first link in the pagination when the carousel is created, as follows:

```
pages.find('a').bind('click', function(){
  carousel.scroll($.jcarousel.intval($(this).text()));
  $(this).siblings('.current').removeClass('current');
  $(this).addClass('current');
  return false;
}).filter(':first').click();
```

Remember that jQuery allows us to chain methods—even though we've got a whole function written inside the `bind()` method, we can still chain the next method to the end of it. We call the `filter()` method to narrow down the list of links to just the first one, then call the `click()` method to fire off the click function we just bound to the link.

Now if you refresh the page in the browser, you'll see that the first link is highlighted with our current class CSS.


1 2 3 4 5 6 7 8 9 10 11 12
Switzerland
Switzerland, officially the Swiss Confederation, is a federal republic consisting of 26 cantons, with Bern as the seat of the federal authorities

Next and previous buttons

Now we've got our slider set up and page numbers working, but we also want to have simple next and previous buttons to make it easy to flip through the slides one at a time. We'll add those at either end of the pagination controls.

Time for action – adding next and previous buttons

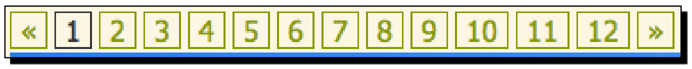
Now all that's left to add is a next and a previous button.

1. We'll add the previous button at the beginning of the pagination, and the next button at the end. Here's how we can use jQuery to insert those links in our document:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }

  pages.find('a').bind('click', function(){
    carousel.scroll($.jcarousel.intval($(this).text()));
    $(this).siblings('.current').removeClass('current');
    $(this).addClass('current');
    return false;
  }).filter(':first').click();

  controls.prepend('<a href="#" id="prev">&laquo;</a>');
  controls.append('<a href="#" id="next">&raquo;</a>');
}
```

I've used the `prepend()` method to insert the previous button before the page numbers and the `append()` method to insert the next button after the page numbers.

If you refresh the page in the browser, you'll see the next and previous buttons show up along with our pagination buttons.



However, clicking them won't cause anything to happen—we have to hook up those buttons so that they work. Let's start with the next button.

2. Just like with the pagination buttons, we need to bind a click event. Again, the jCarousel plugin provides a nice way for us to advance to the next slide.

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }

  pages.find('a').bind('click', function(){
    carousel.scroll($.jcarousel.intval($(this).text()));
    $(this).siblings('.current').removeClass('current');
    $(this).addClass('current');
    return false;
  }).filter(':first').click();

  controls.prepend('<a href="#" id="prev">&laquo;</a>');
  controls.append('<a href="#" id="next">&raquo;</a>');

  $('#next').bind('click', function() {
    carousel.next();
    return false;
  });
}
```

We're selecting the next button and binding a click event. We're canceling the browser's default action so that the browser doesn't attempt to do anything when the link is clicked. Then, all we have to do is call `carousel.next()` and jCarousel will take care of advancing to the next slide for us.

Refresh the page in the browser, and you'll see that clicking the next button advances the slider by one slide.

You'll also notice, however, that the currently highlighted page in the pagination isn't updated. Let's take a look at how we can take care of that.

3. We'll get started by finding the currently highlighted page number as follows:

```
$('#next').bind('click', function() {
  carousel.next();
  var current = pages.find('.current');
  return false;
});
```

Here we're just looking inside our page numbers to find the one with the `current` class.

4. Next, we'll remove the `current` class, move to the next page number link, and add the `current` class to that one as follows:

```
current.removeClass('current').next().addClass('current');
```

Ah, but not so fast, we only want to do that if there is a next link to go to. If there's not, then we don't want to do anything at all. If we check `current.next().length`, we can tell if there's a next link or not. So, we just have to wrap this bit of code in an `if` statement as shown in the following code:

```
if ( current.next().length ) { current.removeClass('current').
next().addClass('current'); }
```

Now if you refresh the page in a browser, you'll see that the next button works as expected. When we get to the last page, it does nothing, just as we'd expect.

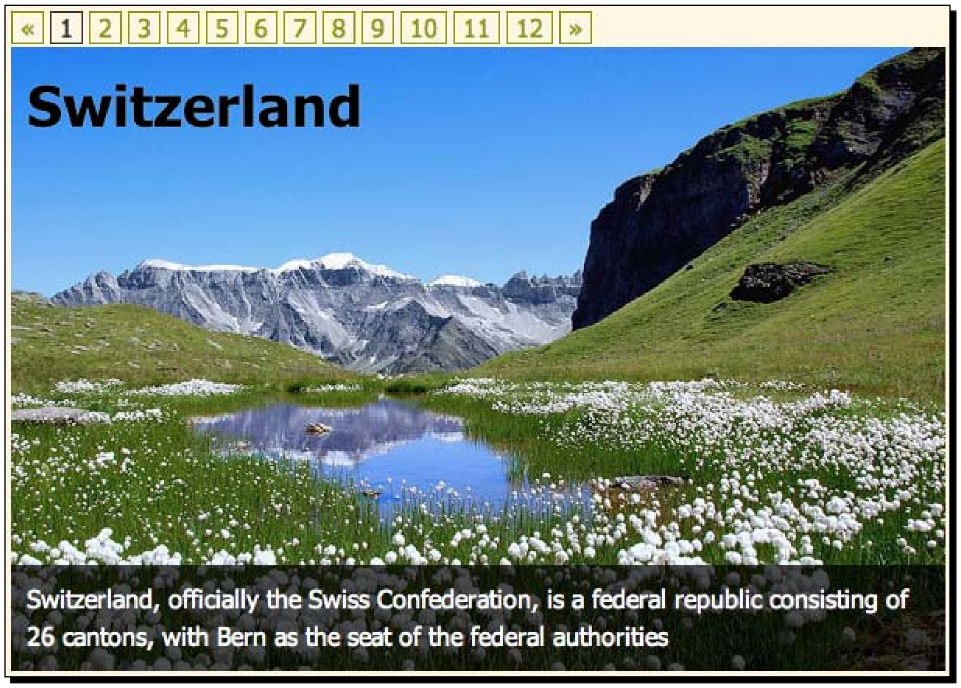
5. Now we'll repeat that whole process with the previous button—the function is very similar. The following is what it will look like:

```
$('#prev').bind('click', function(){
  carousel.prev();
  var current = pages.find('.current');
  if ( current.prev().length ) { current.removeClass('current').
prev().addClass('current'); }
  return false;
});
```

Here's what our complete `carouselInit()` function looks like:

```
function carouselInit(carousel)  {
  var slides = slider.find('li');
  slider.before('<span id="page-controls"><span id="pages"></
span></span>');
  var controls = $('#page-controls');
  var pages = $('#pages');
  for (i=1; i<=slides.length; i++) {
    pages.append('<a href="#">' + i + '</a>');
  }

  pages.find('a').bind('click', function(){
    carousel.scroll($.jcarousel.intval($(this).text()));
    $(this).siblings('.current').removeClass('current');
    $(this).addClass('current');
    return false;
  }).filter(':first').click();

  controls.prepend('<a href="#" id="prev">&laquo;</a>');
  controls.append('<a href="#" id="next">&raquo;</a>');

  $('#prev').bind('click', function(){
    carousel.prev();
    var current = pages.find('.current');
    if ( current.prev().length ) { current.removeClass('current').
prev().addClass('current'); }
    return false;
  });

  $('#next').bind('click', function() {
    carousel.next();
    var current = pages.find('.current');
    if ( current.next().length ) { current.removeClass('current').
next().addClass('current'); }
    return false;
  });
}
```

Now if you refresh the page in a browser, you'll see that the next and previous buttons are both working as expected, along with the page numbers. You can navigate to any slide in the slider by using these external controls.



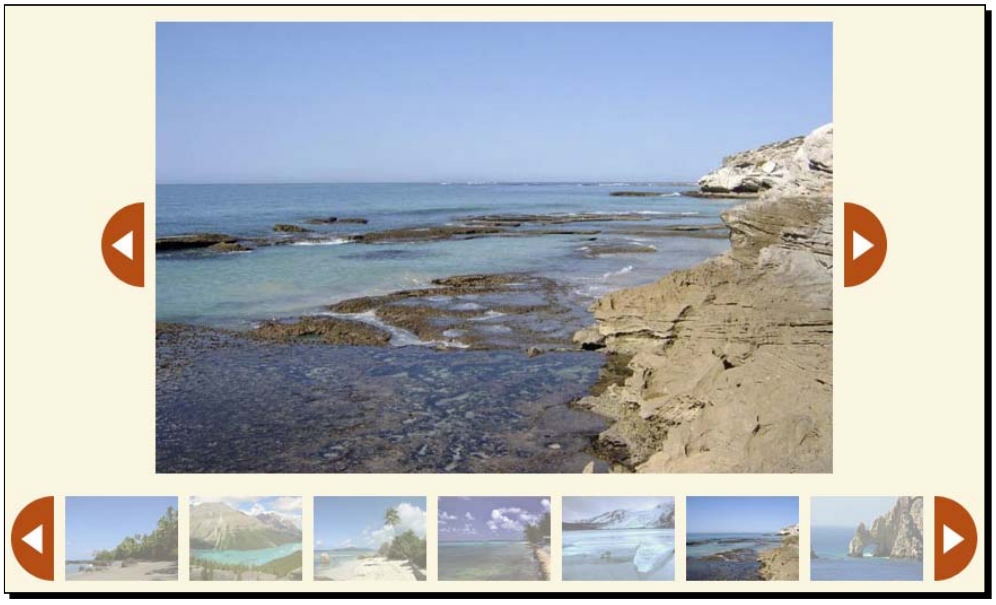
Switzerland, officially the Swiss Confederation, is a federal republic consisting of 26 cantons, with Bern as the seat of the federal authorities

What just happened?

We set up jCarousel to display a single slide at a time. We made sure that jCarousel was not creating its own next and previous buttons. We used jQuery to add next, previous, and pagination buttons to our document, and then used jCarousel's helpful methods to control the carousel from these external controls. We made sure the currently displayed slide is highlighted in the pagination to make it easy for our site visitors to see where they are in the slides.

Carousel slideshow

Now that we've learned how to set up external controls that control the carousel, let's take things the other way as well, and set up our carousel to control a slideshow. In this section, we'll be creating a simple crossfade slideshow that's controlled by a carousel of thumbnail images. The following is a sample of what we'll be creating:



Clicking on any of the thumbnails inside the carousel will load up the large version of that image in the slideshow area. I've also provided next and previous buttons near the slideshow that allow the site visitor to advance one photo at a time through the slideshow without having to click individual thumbnails. Let's take a look at how to put this together.

Time for action – creating a thumbnail slideshow

Setting up the carousel thumbnail slideshow will be the trickiest thing we've done with jCarousel yet. But don't worry, we'll take it one step at a time.

1. I bet you can guess how we're going to get started, can't you? That's right, by setting up our simple HTML file and associated files and folders, just as we did in *Chapter 1*, *Designer, Meet jQuery*. In this case, we want just a simple list of thumbnails that are linked to the full-size version of the image. And we're going to wrap that up in a `<div>` for styling purposes. Here's what my list looks like:

```
<div class="jcarousel-skin-slideshow">
  <ul id="thumb-carousel">
    <li><a href="images/600/Switzerland.jpg"><img src="images/
small/Switzerland.jpg" alt="Switzerland"/></a></li>
    <li><a href="images/600/CostaRica.jpg"><img src="images/small/
CostaRica.jpg" alt="Costa Rica"/></a></li>
    <li><a href="images/600/Canada.jpg"><img src="images/small/
Canada.jpg" alt="Canada"/></a></li>
  ...
  </ul>
</div>
```

   I've got twelve items in my list total, and they're all marked up identically.

2. Next, we'll write the CSS for the carousel. It's a custom design, so we won't be including one of the stylesheets provided with jCarousel. Open up your `styles.css` file and add the following CSS:

```
.jcarousel-skin-slideshow .jcarousel-container  {  }
.jcarousel-skin-slideshow .jcarousel-container-horizontal  {
width:760px;padding:0 48px; }
.jcarousel-skin-slideshow .jcarousel-clip  {  overflow:hidden; }
.jcarousel-skin-slideshow .jcarousel-clip-horizontal  {
width:760px;height:75px; }
.jcarousel-skin-slideshow .jcarousel-item  {
width:100px;height:75px; }
.jcarousel-skin-slideshow .jcarousel-item-horizontal  { margin-
left:0;margin-right:10px; }
.jcarousel-skin-slideshow .jcarousel-item-placeholder  {
background:#fff;color:#000; }
.jcarousel-skin-slideshow .jcarousel-next-horizontal  { position:a
bsolute;top:0;right:0;width:38px;height:75px;cursor:pointer;backgr
ound:transparent url(images/arrow-right.png) no-repeat 0 0; }
.jcarousel-skin-slideshow .jcarousel-next-horizontal:hover,
.jcarousel-skin-slideshow .jcarousel-next-horizontal:focus {
background-position:0 -75px; }
.jcarousel-skin-slideshow .jcarousel-next-horizontal:active {
background-position: 0 -75px; }
.jcarousel-skin-slideshow .jcarousel-prev-horizontal { position:ab
solute;top:0;left:0;width:38px;height:75px;cursor:pointer;backgrou
nd:transparent url(images/arrow-left.png) no-repeat 0 0; }
.jcarousel-skin-slideshow .jcarousel-prev-horizontal:hover,
.jcarousel-skin-slideshow .jcarousel-prev-horizontal:focus {
background-position: 0 -75px; }
.jcarousel-skin-slideshow .jcarousel-prev-horizontal:active {
background-position: 0 -75px; }
```

I've created an image sprite containing the images for my next and previous buttons and that's what's being used as the background image for those. The rest of this should look familiar—setting up the appropriate sizes for the individual items and the carousel itself.

3. Now, we'll attach the jCarousel plugin at the bottom of the document, in between jQuery and your `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.jcarousel.min.js"></script>
<script src="scripts/scripts.js"></script>
```

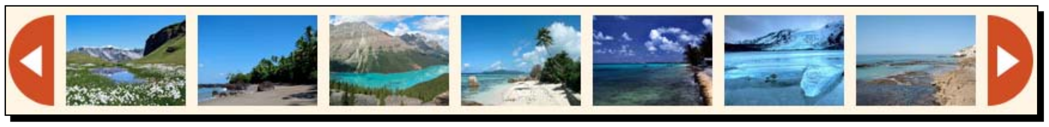
4. Open up your `scripts.js` file and we'll get the JavaScript started by getting our thumbnail carousel up and running. Inside a document ready statement, select the carousel and call the `jcarousel()` method as follows:

```
$(document).ready(function(){
  $('#thumb-carousel').jcarousel({
    scroll: 6,
    wrap: 'circular'
  });
});
```

We've assigned a value of `'circular'` to the `wrap` key—that means the carousel will have neither beginning nor end—it will just continuously wrap around as the site visitor scrolls through.

The continuous wrapping is nice—our site visitors will be able to click either the forward or back carousel buttons no matter where they are, which feels a little friendlier than disabled buttons. However, continuous scrolling can make it a little more difficult for our site visitors to keep track of where they are in the carousel. For that reason, we've set the scroll to `6`, even though our carousel is capable of displaying seven images.

Let's say our site visitor is looking at our carousel and there's a photo of a gorgeous beach scene in the first slot in the carousel. The site visitor clicks the previous button and that gorgeous beach scene slides over to fill the last slot in the carousel. Seeing that same image in a new position helps to communicate what just happened and ensures our site visitors that they didn't miss anything.



What just happened?

We followed steps similar to what we've done in earlier jCarousel examples. Set up our HTML, wrote some CSS styles for the carousel, and then used jQuery to select the list of thumbs and called the `jCarousel()` method. Now, let's get more advanced and add a slideshow to our carousel.

Slideshow

Now that we've got our simple carousel set up and styled the way that we'd like, let's dive into adding the crossfade slideshow feature.

Time for action – adding the slideshow

The jCarousel plugin has taken care of setting up the carousel for us, but we want to get fancy and also add a slideshow area.

1. We're on our own here, so we'll create a separate function for creating the slideshow area. Then we'll call the new function inside our document ready statement:

```
function slideshowInit() {
  // Slideshow setup goes here
}

$(document).ready(function(){
  slideshowInit();
  $('#thumb-carousel').jcarousel({
    scroll: 6,
    wrap: 'circular'
  });
});
```

2. First up, we'll wrap a container around our thumbnail list to create the slideshow area. We find ourselves already in need of referring to the thumbnail list again, so let's store a reference to it in a variable and update the call to the `jcarousel()` method as follows:

```
var thumbs = $('#thumb-carousel');

function slideshowInit() {
  // Slideshow setup goes here
}

$(document).ready(function(){
  slideshowInit();
  thumbs.jcarousel({
    scroll: 6,
    wrap: 'circular'
  });
});
```

3. Next, inside the `slideshowInit()` function, we'll call jQuery's `wrap()` method to wrap the list in a `<div>`.

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
}
```

4. Next, we need to create the actual stage where the full-size images will be featured. We also need to create the next and previous buttons. We're going to use the `prepend()` method so that these elements are inserted into `stage-wrap div` before the thumbs list.

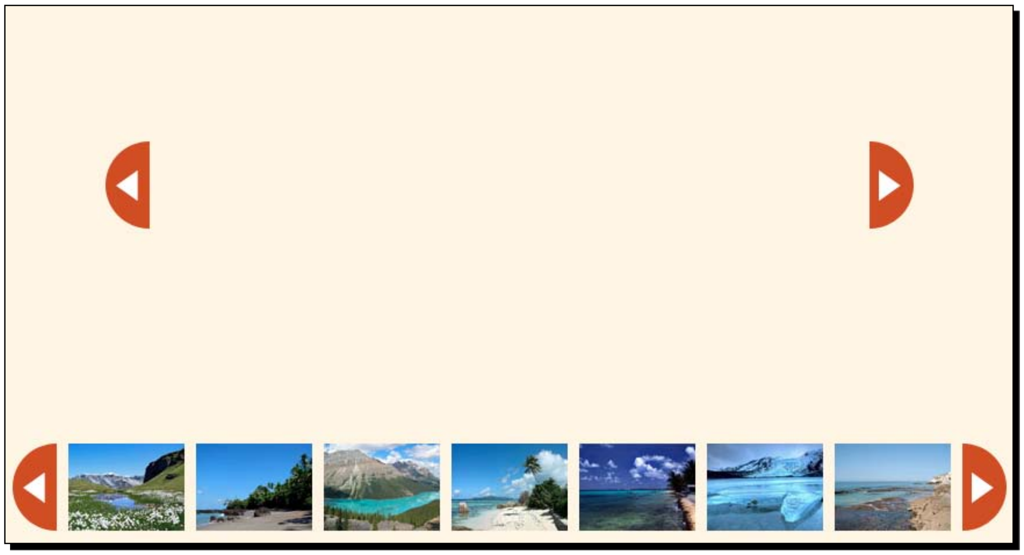
```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
}
```

5. Now, we'll pop back into our `styles.css` file and add some styles for these new elements as follows:

```
#stage-wrap  { position:relative;width:856px; }
#stage  { width:600px;height:400px;padding:0 0 20px
0;position:relative;text-align:center;margin:0 128px; }
#stage img  { position:absolute;top:0;left:50%;margin-left:-300px;
}
```

```
#slideshow-next  { position:absolute;right:80px;top:160px;width:3
8px;height:75px;cursor:pointer;background:transparent url(images/
arrow-right.png) no-repeat 0 0; }
#slideshow-next:hover,
#slideshow-next:active  { background-position:0 -75px; }
#slideshow-prev  { position:absolute;left:80px;top:160px;width:38
px;height:75px;cursor:pointer;background:transparent url(images/
arrow-left.png) no-repeat 0 0; }
#slideshow-prev:hover,
#slideshow-prev:active  { background-position:0 -75px; }
```

All of our full-size images are the same size, 600x400, so we can set that as the width and height of the stage and position the next and previous image buttons accordingly. If you view the page in a browser now, you should see a large blank area left for the stage and the next and previous image buttons on either side of it, all positioned above the thumbnail carousel.



6. We've got a carousel, we've got an empty stage, and we've got next and previous buttons on either side of our stage. Next, we'll populate the stage with an image slideshow. We'll get started by setting up a variable to refer to the stage and setting the `opacity` of the stage to `0` as shown in the following code:

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
```

```
  var stage = $('#stage');
  stage.css('opacity',0);
}
```

We've hidden the stage from view so that we can load the images into it without the site visitor seeing the images loading. This lets us have some control over how the slideshow appears as it's being created. We're going to keep the stage invisible until there's something to see.

7. Next, we'll need to get all the links to the full-size images and get ready to find the URL for each full-size image as follows:

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
  var stage = $('#stage');
  stage.css('opacity',0);
  var imageLinks = thumbs.find('a');
  var src;
}
```

The links to the full-size images are contained in the thumbnail list, which we can refer to with the `thumbs` variable. We're just finding all of the links in that list and storing them in a variable called `imageLinks`. Next, we're setting up an empty container called `src` where we're going to store the url for the images. Though for now, we're leaving that container empty. We'll fill it up in a moment.

8. We've got 12 links to full-size images. For each link, we need to create a new image on the stage. We'll use jQuery's `each()` method to loop through each link and create an image.

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
  var stage = $('#stage');
  stage.css('opacity',0);
  var imageLinks = thumbs.find('a');
  var src;
  imageLinks.each(function(i) {
    // We'll create our images here
  });
}
```

This is the jQuery way of saying *For each link, do this thing*.

9. Next, we'll create an image for each of the links. First up, we know that the `src` attribute of the image is going to equal the `href` attribute of the link. In other words, a link as follows:

```
<a href="images/600/Switzerland.jpg">Switzerland</a>
```

will be used to create an image as follows:

```
<img src="images/600/Switzerland.jpg"/>
```

So the first thing we'll do is get that empty `src` variable we created earlier and store the URL for the image in it:

```
imageLinks.each(function(i) {
  src = $(this).attr('href');
});
```

Next, we're going to create an image with this `src` attribute. I'm going to store my newly created image in a variable called `img`:

```
  imageLinks.each(function(i) {
src = $(this).attr('href');
    var img = $('<img/>', {
      src: src,
      css: {
        display: 'none'
      }
    });
  });
```

We've set the display of the image to none, to hide all of the images created in this way. We've set the `src` attribute of the image to the `src` variable that's holding the URL of the image.

10. Now that the image is created, we'll add it to the stage.

```
imageLinks.each(function(i) {
  src = $(this).attr('href');
  var img = $('<img/>', {
    src: src,
    css: {
      display: 'none'
    }
  });
  img.appendTo(stage);
});
```

jQuery's `appendTo()` method lets us append the image to the stage.

11. Now that the stage is full of images, let's go ahead and make it visible again.

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
  var stage = $('#stage');
  stage.css('opacity',0);
  var imageLinks = thumbs.find('a');
  var src;
  imageLinks.each(function(i) {
    src = $(this).attr('href');
    var img = $('<img/>', {
      src: src,
      css: {
        display: 'none'
      }
    });
    img.appendTo(stage);
  });
  stage.css('opacity',1);
}
```

12. Next, we want to show the appropriate image in the stage when one of the thumbnail links in the carousel is clicked. If you click the thumbnails now, you'll see that it opens the full-size image in the browser, but we want the image to show in the stage instead. We just need a way to reference a particular image in the stage from an image in the carousel. There are several different ways we could go about that—there's nearly always multiple ways to get something done. In this case, we're going to take advantage of jQuery's `data()` method to store an index number in each thumbnail link. I'll then use that index to find and show the appropriate image.

   Basically, we're going to number the links in the list. You'd think they'd be numbered 1 through 12, but remember that JavaScript counting starts at 0, so the thumbnail images will be numbered 0 through 11. When a thumbnail is clicked, we'll get the index number of that thumbnail, find the image on the stage with that same index and show it. So if our site visitor clicks thumbnail number 6, we'll find image number 6 on the stage and show it.

   First up, we have to assign the index numbers to the thumbnails. Inside the document ready statement, add a small function to loop through each thumbnail and add an index number as follows:

```
$(document).ready(function(){
  thumbs.find('a').each(function(index){
    $(this).data('index', (index));
  });
  slideshowInit();
  thumbs.jcarousel({
    scroll: 6,
    wrap: 'circular',
    initCallback: nextPrev
  });
});
```

13. Now that all of the thumbnail links are numbered, we can write a function that will find the appropriate image on the stage and show it when the thumbnail is clicked. Inside of the `slideshowInit()` function, we'll bind our function to the click event:

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
  var stage = $('#stage');
  stage.css('opacity',0);
  var imageLinks = thumbs.find('a');
  var src;
  imageLinks.each(function(i) {
    src = $(this).attr('href');
    var img = $('<img/>', {
      src: src,
      css: {
        display: 'none'
      }
    });
    img.appendTo(stage);
  });
  stage.css('opacity',1);
  imageLinks.bind('click', function(){
    // Function to find and show an image goes here
  });
}
```

14. The first thing to do inside our new function is to cancel the browser's default behavior. We don't want the link to open the image in the browser, so we'll return false.

```
imageLinks.bind('click', function(){
  return false;
})
```

15. Next, we need to get the number that we stored in our link. We'll use the `data()` method again to find the number:

```
imageLinks.bind('click', function(){
  var index = $(this).data('index');
  return false;
})
```

16. Now, we need to search in the stage for the image with that index number. I'm going to store the image in a variable called `nextImage` since it will be the next image to show.

```
imageLinks.bind('click', function(){
  var index = $(this).data('index');
  var nextImage = stage.find('img:eq(' + index + ')');
})
```

jQuery allows us to find an element by its index number using the `:eq` selector. For example, the `$('img:eq(1)')` selector would select the second image in a list of images. (Remember, JavaScript counting starts at 0 instead of 1.) In this case, I know which number image I want because it's the number stored in the link that was just clicked.

17. Now that we've got the next image, we need to show it. We're going to fade it in and add a class of `active` to it.

```
imageLinks.bind('click', function(){
  var index = $(this).data('index');
  var nextImage = stage.find('img:eq(' + index + ')');
  nextImage.fadeIn().addClass('active');
  return false;
})
```

18. But don't forget that there's already another image visible. We need to find that one and fade it out. Since we're adding a class of `active` when the image is shown, we can easily find the currently displayed image by looking for the one with the class of `active`:

```
imageLinks.bind('click', function(){
  var index = $(this).data('index');
  var nextImage = stage.find('img:eq(' + index + ')');
  stage.find('img.active').fadeOut().removeClass('.active');
  nextImage.fadeIn().addClass('active');
  return false;
})
```

Don't forget that we'll have to be sure to remove that `active` class so that only one image will be marked active at a time.

If you refresh the page in the browser now, you'll see that clicking one of the thumbnail links in the carousel loads up the corresponding image in the slideshow. One image fades out while the next image fades in, in a nice smooth manner. Next, we'll get those next and previous buttons working so that we can use them to easily flip from one image to the next.

What just happened?

Phew! I hope you're still with me because this is a pretty awesome way to present a slideshow of images to your site visitors. I hope that you're starting to see that sometimes a plugin can be simply a beginning—you can get creative and invent your own functionality to layer on top of the default plugin behavior.

Next and previous buttons

We're definitely making some nice progress. Clicking the thumbnails loads up the full-size version of the image in the slideshow, and we can use the carousel controls to scroll through the thumbnails and see them all. Now, let's get the next and previous image buttons working.

Time for action – activating the Next and Previous Buttons

Next up, we'll get those next and previous buttons around the image working so that the site visitor can easily flip through all the images.

1. Just like when we hooked up external controls to the carousel in the last example, we'll get started by setting up a callback function for the carousel. We'll call the function `nextPrev` and set it up as follows:

   ```
   function nextPrev(carousel) {
   }

   thumbs.jcarousel({
     scroll: 6,
     wrap: 'circular',
     initCallback: nextPrev
   });
   ```

   Now the `nextPrev` function will be called when the carousel is initialized.

2. Inside the `nextPrev()` function, we'll select the previous button and bind a function to the click event:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    //Click code will go here
  });
}
```

3. When a site visitor clicks the previous button, we want to show the previous image in the slideshow. As usual with JavaScript, there's more than one way to go about that. Since we've already got a nice slide switch set up to happen when one of the thumbnails in the carousel is clicked, let's just go ahead and re-use that.

   When our site visitor clicks the previous button, we'll find the previous thumbnail in the carousel and click it. That will kick off the image transition and allow us to re-use the code we've already written.

   So our first order of business is to find the currently selected thumbnail. However, we haven't made it easy to find the current thumbnail. So let's go back inside our `slideshowInit()` function and add a line of code to add a class to the current thumbnail:

```
function slideshowInit() {
  thumbs.wrap('<div id="stage-wrap"></div>');
  $('#stage-wrap').prepend('<div id="slideshow-next"></div><div
id="slideshow-prev"></div><div id="stage"></div>');
  var stage = $('#stage');
  stage.css('opacity',0);
  var imageLinks = thumbs.find('a');
  var src;
  imageLinks.each(function(i) {
    src = $(this).attr('href');
    var img = $('<img/>', {
      src: src,
      css: {
        display: 'none'
      }
    });
    img.appendTo(stage);
  });
  stage.css('opacity',1);
  imageLinks.bind('click', function(){
    var index = $(this).data('index');
    $(this).parents('li').addClass('current').siblings('.
current').removeClass('current');
    var nextImage = stage.find('img:eq(' + index + ')');
    stage.find('img.active').fadeOut().removeClass('.active');
    nextImage.fadeIn().addClass('active');
    return false;
  })

}
```

Here, we're adding a class of `current` to the `<li>` tag that contains the clicked thumbnail. Then we're checking all the siblings to remove the `current` class if it exists somewhere else. This ensures that only one item in the carousel will have the `current` class at any given time.

4. Now, if you'll humor me for a minute, we'll take a sidetrip to the CSS. Since we're adding a class to the current thumbnail, we can make use of that for CSS purposes to style the current thumbnail differently than the rest. Let's reduce the opacity of the thumbnails and make the current one 100 percent opaque to make it stand out. Open up `styles.css` and add some styles for this as follows:

```
#thumb-carousel img         { opacity:.5; }
#thumb-carousel .current img  { opacity:1; }
```

5. Back to the JavaScript! Now that we've got an easy way to select the current thumbnail, we just have to find the one with the `current` class. Inside the `prevNext()` function, we can get the current link this way:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
  });
}
```

6. Since this is the function attached to the previous button, we'll need to find the previous thumbnail in the list. I'll use jQuery's `prev()` method to find the previous thumbnail in the carousel:

```
currentSlide.prev();
```

However, if the current slide is the first one, there isn't a previous slide to go to. In this case, if the site visitor is on the first slide and clicks the previous button, I want them to go to the last slide in the list so that it continues seamlessly. So, I'll first check to see if there is a previous slide as follows:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length ? currentSlide.
prev() : thumbs.find('li:last');
  });
}
```

There are a couple of things to explain here. First, this line, translated into English from JavaScript, says *Is there a thumbnail before this one? If there is, then that's where we're going. If there's not, then we're heading over to the last thumbnail*.

The second thing to explain is a new way of writing this that we haven't seen before. This is called a ternary operator and it's a shorthand way of writing this:

```
var prevSlide;
if (currentSlide.prev().length) {
  prevSlide = currentSlide.prev();
} else {
  prevSlide = thumbs.find('li:last');
}
```

Here's how a ternary operator works:

```
condition to check ? value if true : value if false
```

It starts with the condition that we're checking which is followed by a `?`. After that, we have the value if that condition is true followed by a `:`, and the value if the condition is false.

7. Now that we've found the previous slide, all that's left to do is click the link inside as follows:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length? currentSlide.
prev() : thumbs.find('li:last');
    prevSlide.find('a').click();
  });
}
```

This will fire off the function we've written to change the slide in the browser. If you reload the page in the browser at this point and click the previous button a few times, you'll see that the image switches just as we'd expect.

However, there's not much going on with the carousel. It's just sitting there. And right away the currently selected thumbnail is out of view. If I click the previous button once, then scroll the carousel, I can finally see the highlighted thumbnail. Ideally, the carousel would update itself to be sure that the current thumbnail was always visible.

8. The jCarousel plugin makes it easy for us to scroll to any slide in the carousel. We only have to know which one we want to show. A part of the jCarousel's setup script also assigns a `jcarouselindex` attribute to each list item in the carousel. We can get that number and use it for scrolling purposes. First, let's figure out what `jcarouselindex` of the `prevSlide` is, since that's where we want to scroll.

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length? currentSlide.
prev() : thumbs.find('li:last');
    var index = parseInt(prevSlide.attr('jcarouselindex'));
    prevSlide.find('a').click();
  });
}
```

I'm using `parseInt()` to make sure that I get a number instead of a string. If I get a string back, it can mess up the scrolling in the carousel.

Now, all that's left to do is scroll to the right thumbnail:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length? currentSlide.
prev() : thumbs.find('li:last');
    var index = parseInt(prevSlide.attr('jcarouselindex'));
    prevSlide.find('a').click();
    carousel.scroll(index);
  });
}
```

Now if you refresh the page in the browser, you'll see that clicking the previous button updates the carousel—the carousel will scroll so that the currently highlighted slide is the first one in the carousel. However, what if I decide I want the currently highlighted slide to appear in the middle? Easy! I've got seven slides showing. If the highlighted slide is in the middle, that means there will be three slides before it (and three slides after it). All I have to do is tell the carousel to make the slide three before the highlighted slide the first slide visible as follows:

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length? currentSlide.
prev() : thumbs.find('li:last');
    var index = parseInt(prevSlide.attr('jcarouselindex')) - 3;
```

```
      prevSlide.find('a').click();
      carousel.scroll(index);
    });
  }
```

Now, for example, when I click the previous button, if the next slide is slide number 5, slide number 2 will be shown first in the carousel, which means slide number 5 will be right in the middle of the carousel. Refresh the page in the browser and give it a try. Nice, right?

9. All that's left to do is get the next button working as well as the previous one. The function is almost identical with a few tweaks made.

```
function nextPrev(carousel) {
  $('#slideshow-prev').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var prevSlide = currentSlide.prev().length? currentSlide.
prev() : thumbs.find('li:last');
    var index = parseInt(prevSlide.attr('jcarouselindex')) - 3;
    prevSlide.find('a').click();
    carousel.scroll(index);
  });
  $('#slideshow-next').bind('click', function() {
    var currentSlide = thumbs.find('li.current');
    var nextSlide = currentSlide.next().length ? currentSlide.
next() : thumbs.find('li:first');
    var index = parseInt(nextSlide.attr('jcarouselindex')) - 3;
    nextSlide.find('a').click();
    carousel.scroll(index);
  });
}
```

I'm using the `next()` method instead of the `prev()` method to get the next slide rather than the previous one. Aside from that, the function is the same.

Now if you refresh the page in the browser, you'll see that the next and previous image buttons both work—they show the correct image in the slideshow and scroll the carousel so that the current image is highlighted right in the middle of the carousel.

What just happened?

We combined some external carousel controls with a slideshow to create a robust slideshow/carousel combination. The slideshow can be controlled from the carousel—clicking a thumbnail in the carousel will load up the full-size version of the image in the slideshow stage. And clicking the next and previous buttons in the stage will update the carousel, scrolling it so that the currently highlighted thumbnail appears in the middle of the carousel's viewable area.

We started with some basic HTML, wrote a custom CSS skin for the carousel, and called the `jcarousel()` method to get the carousel working. Next, we wrote a function to dynamically create the slideshow stage and buttons. Finally, we made it all work together with some fancy jQuery footwork.

Summary

We took a look at using the jCarousel plugin in a variety of situations—we created a simple horizontal thumbnail carousel, a vertical news ticker, a featured content slider with external controls, and finally, a carousel/slideshow combo that really showed off the capabilities of the jCarousel plugin. Now you've added another powerful tool to your toolbox—the jCarousel plugin is flexible, powerful, and can be customized to work in a variety of different situations.

Next up, we'll take a look at creating an interactive data grid.

11
Creating an Interactive Data Grid

While you might not consider a data grid to be all that exciting, they do offer a way for site visitors to interact with large amounts of data and understand it in a way they might not be able to otherwise. One of the most exciting developments in HTML5 is the introduction of a grid element, which allows us to easily create an interactive data grid using only markup. However, it's one of the new elements for which browser support is lagging—there is little, if any, browser support for the time being, and it could be years before we're able to make use of this new element. Luckily, we can use jQuery to fill in the gap until the new grid element is ready for primetime.

In this chapter, we'll learn the following topics:

- Turning an ordinary table into an interactive data grid using the DataTables jQuery plugin from Allan Jardine
- Customizing the appearance and behavior of the data grid with help from the jQuery UI Themeroller

Basic data grid

We'll get started by using the DataTables plugin to create a basic data grid, keeping the default settings and the styles provided with the data grid. Data grids are most helpful when we have large amounts of data to present, and site visitors might want to filter and sort the data in different ways to find the information they are looking for. Think, for example, of a list of flights—one site visitor might be interested in sorting the flights by departure time to find the earliest possible departure, while another site visitor might want to sort the flights by duration to find the shortest possible flight.

Presenting the data in an interactive data grid allows each site visitor to quickly and easily find just the information they're looking for in a sea of information. For site visitors with JavaScript disabled, they'll simply see a large table of data and will never know that they're missing out on the interactive features. All of the information will still be available to them.

Time for action – creating a basic data grid

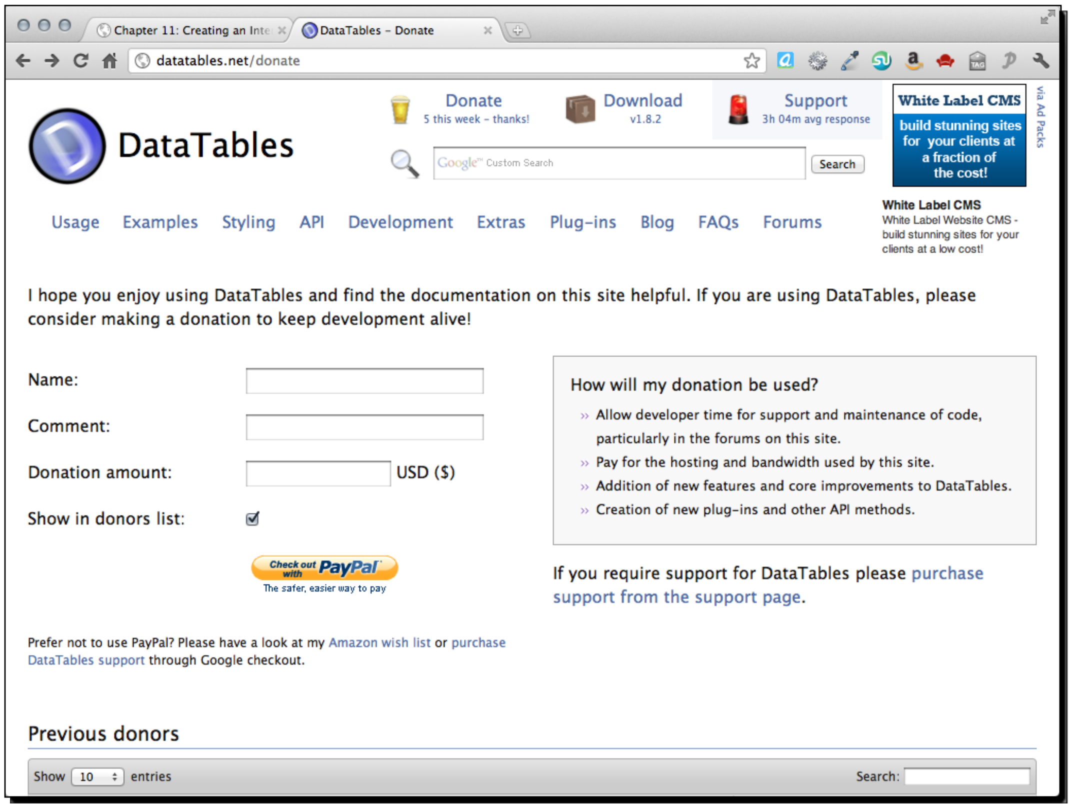
Let's take a look at how to turn a basic HTML table into an interactive data grid:

1. We'll get started as usual with our basic HTML file and associated files and folders, just like we did in *Chapter 1*, *Designer, Meet jQuery*. We'll fill the `<body>` of our HTML document with the HTML markup for a large table of data. The DataTables plugin does require that we are careful and correct with our table markup. We'll need to be sure to use a `<thead>` element for the table's header, and a `<tbody>` element for the table's body. A `<tfoot>` element for the table footer is optional. Here's an abbreviated sample of the HTML markup for a table of the all-time best-selling books:

```
<table id="book-grid">
  <thead>
    <tr>
      <th>Title</th>
      <th>Author</th>
      <th>Original Language</th>
      <th>First Published</th>
      <th>Approximate Sales</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>A Tale of Two Cities</td>
      <td>Charles Dickens</td>
      <td>English</td>
      <td>1859</td>
      <td>200 million</td>
    </tr>
    <tr>
      <td>Le Petit Prince (The Little Prince)</td>
      <td>Antoine de Saint-Exup&eacute;ry</td>
      <td>French</td>
      <td>1943</td>
      <td>200 million</td>
    </tr>
    ...
  </tbody>
</table>
```

I've added a total of 106 books to the table, each marked up just as these are. Note that we've added an `id` of `book-grid` to the table element, and have used the `<th>` elements for the heading of each column, and enclosed those in a `<thead>` element. We've also used a `<tbody>` element to wrap all the rows in the table's body.

2. Next, we'll download the DataTables plugin. Head over to `http://datatables.net` where you'll find the plugin's downloads, documentation, and examples. Click on the **Download** link in the header to download a ZIP file.



I hope you enjoy using DataTables and find the documentation on this site helpful. If you are using DataTables, please consider making a donation to keep development alive!

Name:

Comment:

Donation amount: USD ($)

Show in donors list: ✓

How will my donation be used?

- Allow developer time for support and maintenance of code, particularly in the forums on this site.
- Pay for the hosting and bandwidth used by this site.
- Addition of new features and core improvements to DataTables.
- Creation of new plug-ins and other API methods.

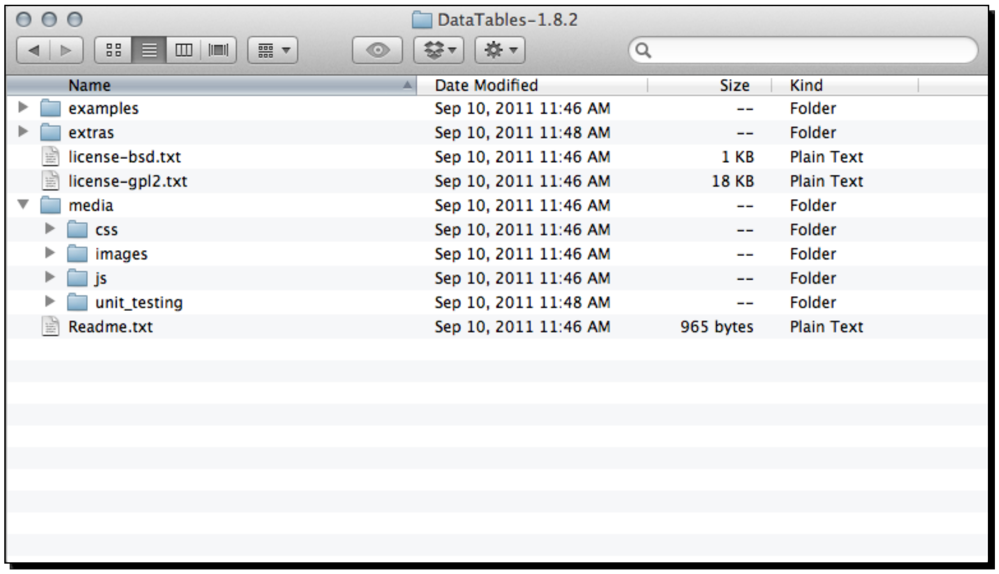
If you require support for DataTables please purchase support from the support page.

Prefer not to use PayPal? Please have a look at my Amazon wish list or purchase DataTables support through Google checkout.

Previous donors

Show 10 entries Search:

3. Unzip the folder and take a look inside.



Name	Date Modified	Size	Kind
examples	Sep 10, 2011 11:46 AM	--	Folder
extras	Sep 10, 2011 11:48 AM	--	Folder
license-bsd.txt	Sep 10, 2011 11:46 AM	1 KB	Plain Text
license-gpl2.txt	Sep 10, 2011 11:46 AM	18 KB	Plain Text
media	Sep 10, 2011 11:46 AM	--	Folder
css	Sep 10, 2011 11:46 AM	--	Folder
images	Sep 10, 2011 11:46 AM	--	Folder
js	Sep 10, 2011 11:46 AM	--	Folder
unit_testing	Sep 10, 2011 11:48 AM	--	Folder
Readme.txt	Sep 10, 2011 11:46 AM	965 bytes	Plain Text

There's a folder of `examples` with several different examples of the DataTables plugin in action. A folder of `extras` provides extra functionality for advanced data tables—we won't be using any of those here. There's a `media` folder that contains `images`, `css`, `js`, and `unit_testing` resources. And finally a `Readme.txt` file contains information on the plugin's creator and where to find the documentation, and so on.

Finally, you'll find the licenses for the plugin, both BSD and GPL. You can read these license files or visit Wikipedia to get the details on these licenses, but they're both free software licenses allowing you to make use of the plugin code for free.

4. We're going to set up a basic example, so we'll just need a couple of things for our own project. First, copy the contents of the `images` directory to your own `images` directory. Open the `css` folder and copy `demo_table.css` to your own `styles` directory. Be careful to select the proper CSS file—`demo_table.css`—because there are a few CSS files in there. Finally, in the `js` folder, find the minified version of the plugin, `jquery.dataTables.min.js`, and copy that to your own `scripts` directory.

5. Next, we'll get all the necessary files attached to our HTML page that contains our table. In the `<head>` section of the document, attach the CSS file, before your own `styles.css` file:

```
<link rel="stylesheet" href="styles/demo_table.css"/>
<link rel="stylesheet" href="styles/styles.css"/>
```

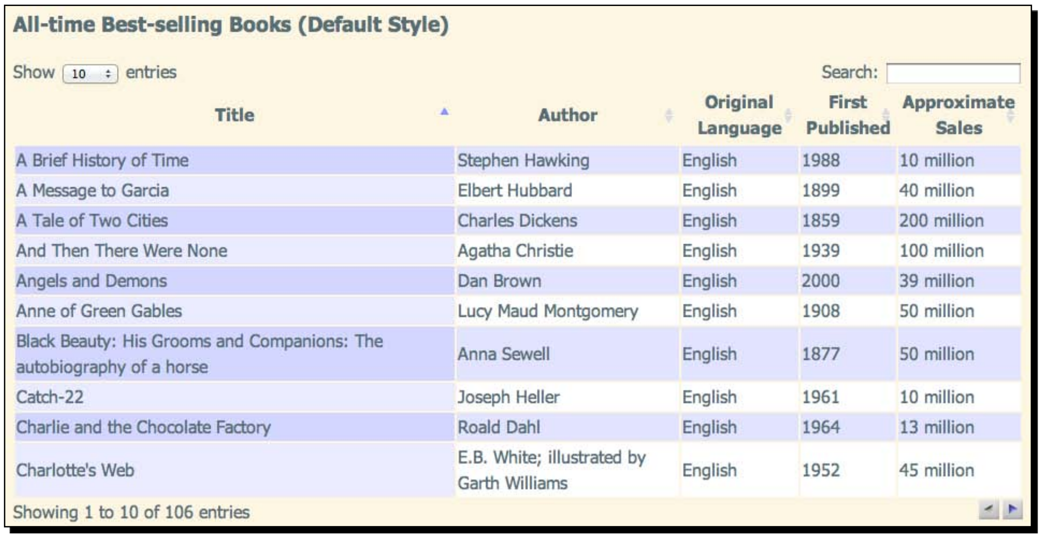
6. Next, at the bottom of the HTML document, attach the DataTables plugin in between jQuery and your own `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.dataTables.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
</html>
```

7. Next, open your `scripts.js` file and inside a document ready statement, select the table and call the `dataTable()` method as follows:

```
$(document).ready(function(){
  $('#book-grid').dataTable();
});
```

Now, if you refresh the page in the browser, you'll see that your table has been transformed into a data grid. You can select how many items to view at one time, type into the search box to find specific table entries, and use the pagination controls at the bottom right to page through the data table's rows.


All-time Best-selling Books (Default Style)

Show 10 entries   Search:

Title	Author	Original Language	First Published	Approximate Sales
A Brief History of Time	Stephen Hawking	English	1988	10 million
A Message to Garcia	Elbert Hubbard	English	1899	40 million
A Tale of Two Cities	Charles Dickens	English	1859	200 million
And Then There Were None	Agatha Christie	English	1939	100 million
Angels and Demons	Dan Brown	English	2000	39 million
Anne of Green Gables	Lucy Maud Montgomery	English	1908	50 million
Black Beauty: His Grooms and Companions: The autobiography of a horse	Anna Sewell	English	1877	50 million
Catch-22	Joseph Heller	English	1961	10 million
Charlie and the Chocolate Factory	Roald Dahl	English	1964	13 million
Charlotte's Web	E.B. White; illustrated by Garth Williams	English	1952	45 million

Showing 1 to 10 of 106 entries

What just happened?

We set up a basic HTML table and turned it into an interactive data grid by attaching some CSS and the DataTables plugin. We selected the table and called the `dataTable()` method to activate the DataTables plugin.

That was pretty easy, wasn't it? Of course, chances are this lavender design doesn't fit the design of your site, so let's take a look at how we can customize the appearance of the data table.

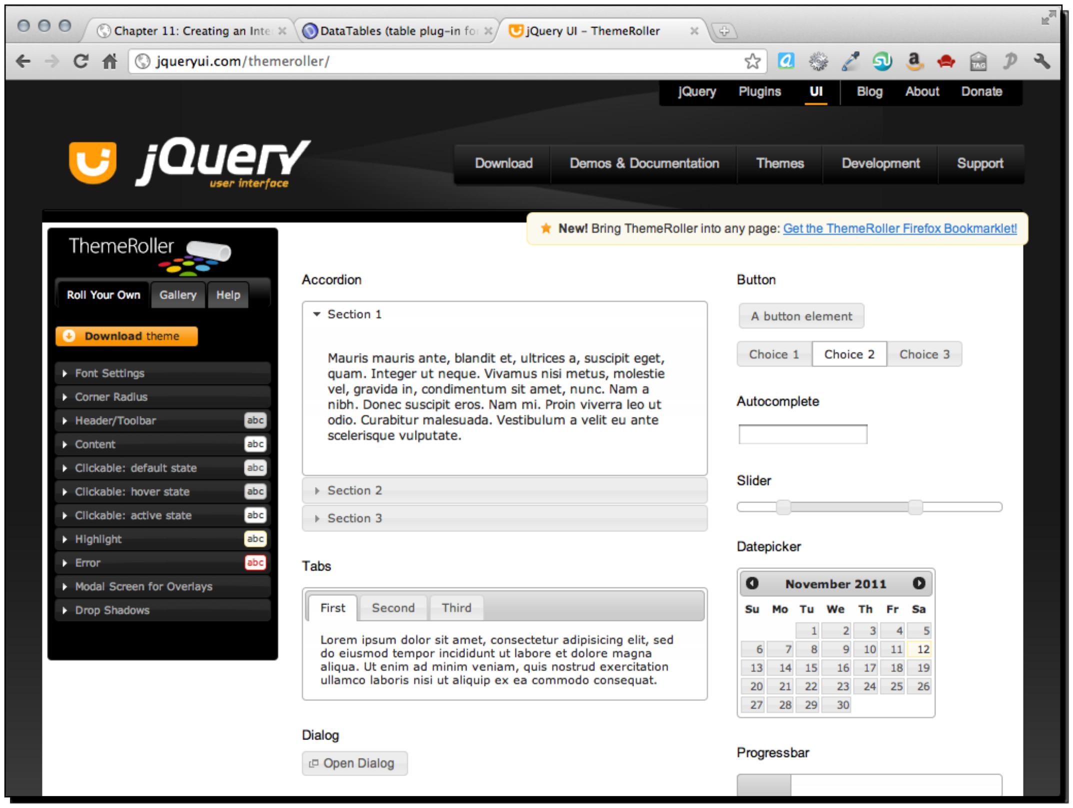
Customizing the data grid

The DataTables plugin is the first plugin we've used that has support for the jQuery UI Themeroller. jQuery UI is a collection of widgets and interactions that make building complex applications easier and faster. Learning jQuery UI itself is beyond the scope of this book, but we'll take a look at how to use the jQuery UI Themeroller to create a custom theme for our data table. This same theme would apply to any jQuery UI widgets used on our page, as well as any jQuery plugins being used that include support for the jQuery UI Themeroller.

Time for action – customizing the data grid

We'll pick up right where we left off with our data table. If you'd like to save your basic example, just save a copy of the file. Then follow these steps to customize the appearance of your data grid:

1. Head over to `http://jqueryui.com/themeroller` where we'll take a look at the Themeroller. In the left column, you'll find the controls for selecting a predefined theme or creating a custom theme, and the wide right column contains samples of several different types of widgets.



New! Bring ThemeRoller into any page: Get the ThemeRoller Firefox Bookmarklet!

Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

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.

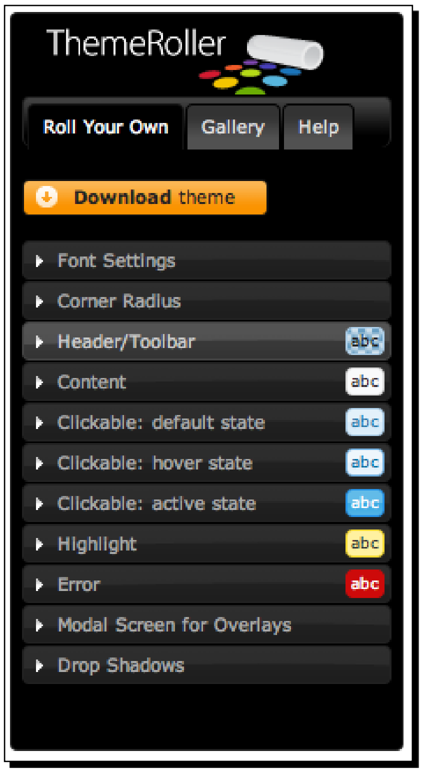
Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

2. Click on the **Gallery** tab in the left column, and you'll see that you have dozens of choices of pre-built Themeroller themes to choose from. As you click on different samples, you'll see the sample widgets in the right column update to reflect that style. I usually like to get started by selecting a prebuilt theme that's reasonably close to the color scheme or appearance that I want, then I flip to the **Roll Your Own** tab to tweak it to suit my needs. For this example, I'm going to start with the **Cupertino** style.

After flipping to the **Roll Your Own** tab, you'll see that there are settings for fonts, colors, corners, headers, and so on. Make whatever adjustments you'd like to get the theme looking just the way you'd like. Feel free to play and experiment. If you go too far and get to something you don't like, it's easy to flip back to the **Gallery** tab and select the prebuilt theme again, stripped of any of your customizations, then start again.

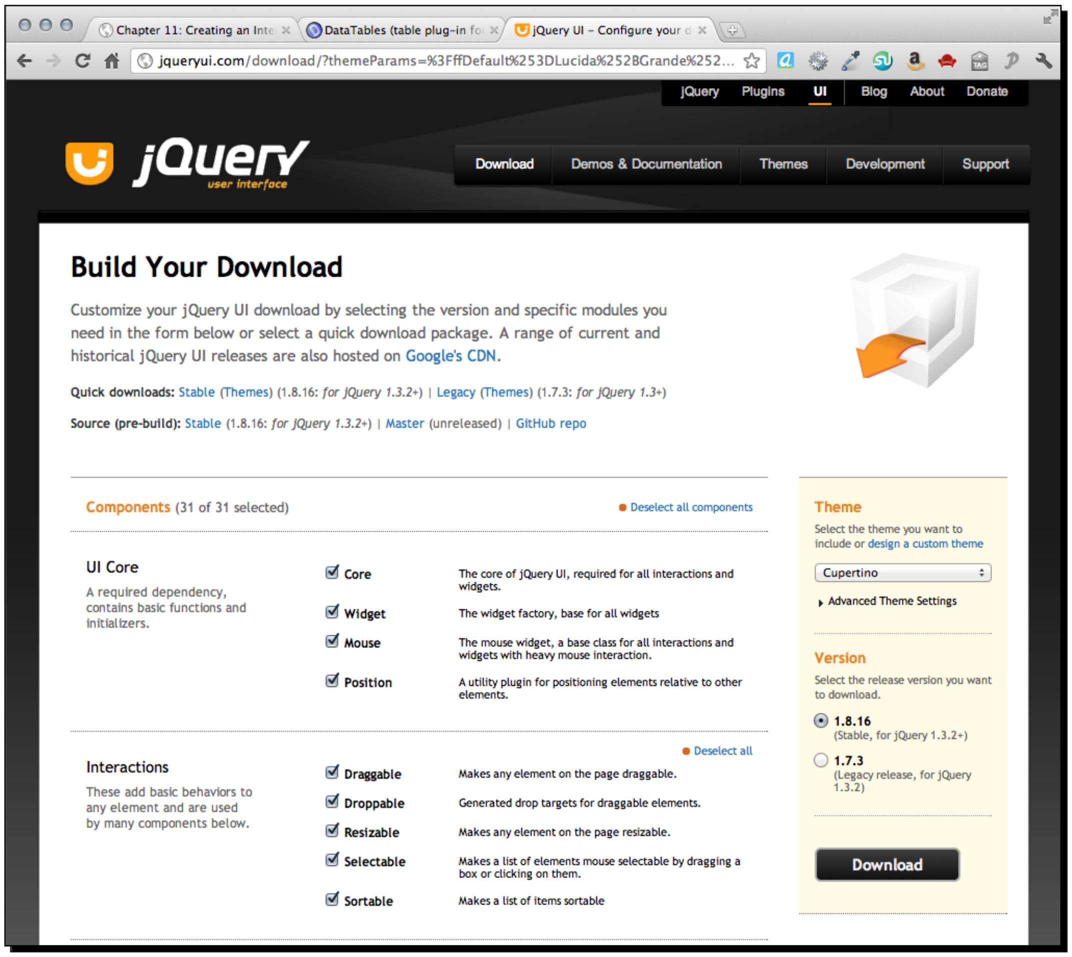
Keep in mind that any of your customizations will be lost if you re-select a prebuilt theme. Once you get something you like, be sure to move on to step 3 to save it.

3. Once you've got your theme set up just the way you'd like, click on the **Download Theme** button.



4. You'll find yourself on the **Build Your Download** page, which might seem a little confusing. Note that jQuery UI is so large and has so many different features on offer, the developers realize that forcing everyone to download the entire thing would be overkill. If you only wanted to use one widget, there'd be no need for downloading all the other widgets and effects. This page lets you pick and choose different components of jQuery UI so that you don't have to download more than you need.

Since we're only here for a theme, we can go ahead and click on the **Deselect all components** link near the top of the page.


Chapter 11: Creating an Inte
DataTables (table plug-in fo
jQuery UI - Configure your d
jqueryui.com/download/?themeParams=%3FffDefault%253DLucida%252BGrande%252...
jQuery
Plugins
UI
Blog
About
Donate
jQuery user interface
Download
Demos & Documentation
Themes
Development
Support

Build Your Download

Customize your jQuery UI download by selecting the version and specific modules you need in the form below or select a quick download package. A range of current and historical jQuery UI releases are also hosted on Google's CDN.

Quick downloads: Stable (Themes) (1.8.16: *for jQuery 1.3.2+*) | Legacy (Themes) (1.7.3: *for jQuery 1.3+*)

Source (pre-build): Stable (1.8.16: *for jQuery 1.3.2+*) | Master (unreleased) | GitHub repo

Components (31 of 31 selected) — Deselect all components

UI Core
A required dependency, contains basic functions and initializers.

☑ Core	The core of jQuery UI, required for all interactions and widgets.
☑ Widget	The widget factory, base for all widgets
☑ Mouse	The mouse widget, a base class for all interactions and widgets with heavy mouse interaction.
☑ Position	A utility plugin for positioning elements relative to other elements.

Deselect all

Interactions
These add basic behaviors to any element and are used by many components below.

☑ Draggable	Makes any element on the page draggable.
☑ Droppable	Generated drop targets for draggable elements.
☑ Resizable	Makes any element on the page resizable.
☑ Selectable	Makes a list of elements mouse selectable by dragging a box or clicking on them.
☑ Sortable	Makes a list of items sortable

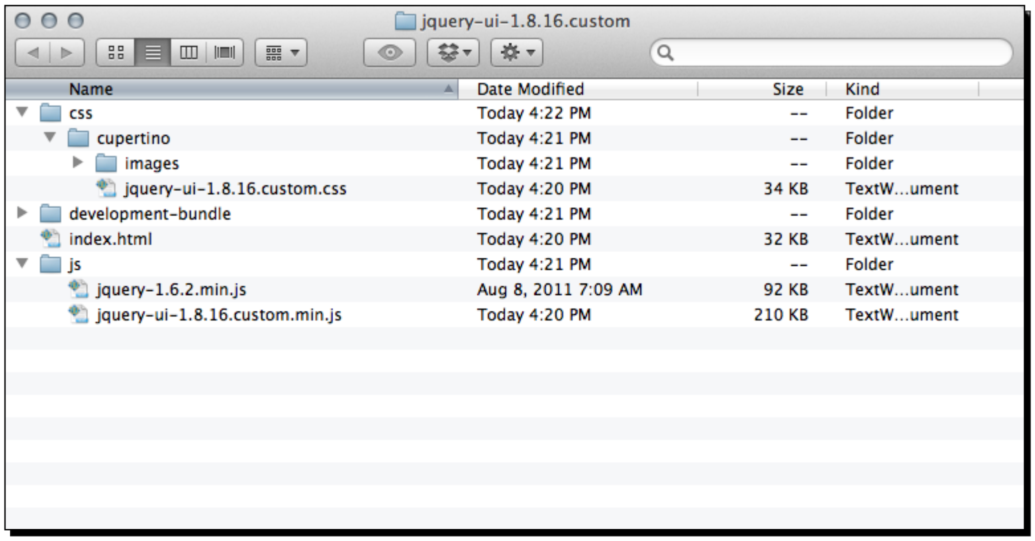
Theme
Select the theme you want to include or design a custom theme
Cupertino
Advanced Theme Settings

Version
Select the release version you want to download.
● 1.8.16 (Stable, for jQuery 1.3.2+)
○ 1.7.3 (Legacy release, for jQuery 1.3.2)

Download

Then, we'll leave the **Theme** and **Version** settings at their defaults and click on the **Download** button to download a ZIP file.

5. Unzip the file and take a look inside. You'll see that even though we got the simplest download we could, we still have quite a few files.


jquery-ui-1.8.16.custom

Name	Date Modified	Size	Kind
css	Today 4:22 PM	--	Folder
cupertino	Today 4:21 PM	--	Folder
images	Today 4:21 PM	--	Folder
jquery-ui-1.8.16.custom.css	Today 4:20 PM	34 KB	TextW...ument
development-bundle	Today 4:21 PM	--	Folder
index.html	Today 4:20 PM	32 KB	TextW...ument
js	Today 4:21 PM	--	Folder
jquery-1.6.2.min.js	Aug 8, 2011 7:09 AM	92 KB	TextW...ument
jquery-ui-1.8.16.custom.min.js	Today 4:20 PM	210 KB	TextW...ument

We've got a `css` folder that contains our theme folder with a CSS file and **images** inside it. We've also got a `development-bundle` folder, an HTML file, and a `js` folder that contains jQuery and a jQuery UI file.

All we need out of all of this is our theme. Copy your theme folder to the `styles` directory of your own project. My theme folder is named `cupertino` since that's the theme I chose. If you selected a different theme, your theme folder will be called something else. It will be easy to find, though, as it will be the only folder inside the `css` folder.

6. Next, we'll attach our theme CSS file to our HTML file. Inside the `<head>` section, attach your theme CSS file before the `demo_table.css` file we attached in the previous example.

```
<link rel="stylesheet" href="styles/cupertino/jquery-ui-1.8.16.
custom.css"/>
<link rel="stylesheet" href="styles/demo_table.css"/>
```

7. Now, unfortunately, our theme CSS file doesn't quite have all the styles we'll need for a nicely styled data grid. After all, the jQuery UI developers have no way of knowing all the different types of widgets and plugins people will want to use, so there's no possible way they could cover every single case. Luckily, DataTables plugin author Allan Jardine has already done some nice work for us in this area and has provided a CSS file with the styles we'll need to get our themed data grid looking its best.

You can read up on styling the DataTables plugin in the documentation Allan Jardine has made available at `http://datatables.net/styling/`.

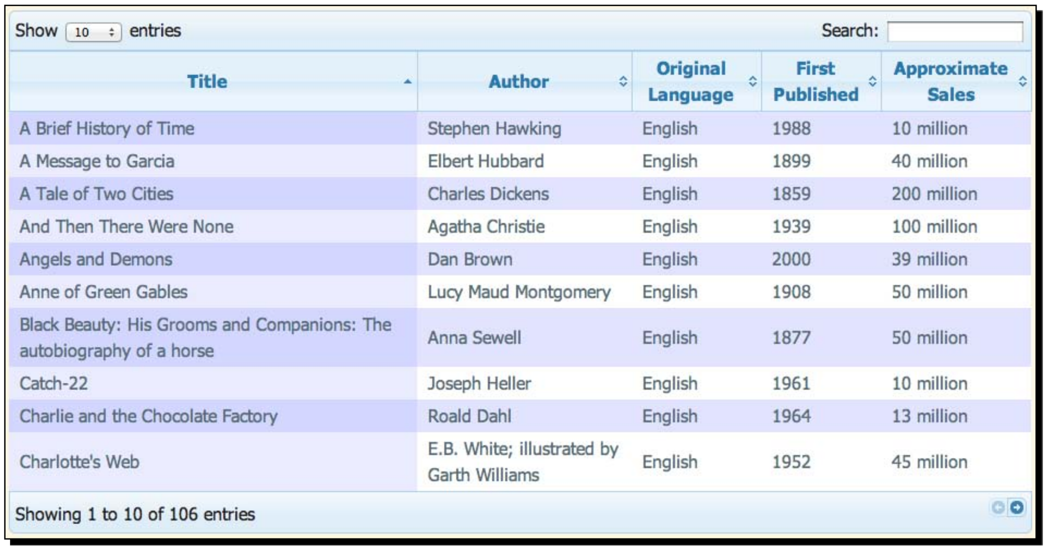
Back inside the DataTables plugin files, look inside the `css` folder inside the `media` folder to find the `demo_table_jui.css` file. Copy that to your own styles folder and update your `<link>` tag to link to this version of the `demo_table.css` instead as follows:

```
<link rel="stylesheet" href="styles/cupertino/jquery-ui-1.8.16.
custom.css"/>
<link rel="stylesheet" href="styles/demo_table_jui.css"/>
```

8. Now we just have to make a small update to our JavaScript code. We have to tell the `dataTable()` method that we want to use jQuery UI. Head back into your `scripts.js` file and we'll add a pair of curly brackets and pass a key/value pair to enable jQuery UI styling for our data table:

```
$(document).ready(function(){
  $('#book-grid').dataTable({
    'bJQueryUI': true
  });
});
```

If you refresh the page in the browser now, you'll see that the data grid is now using a style that's consistent with the widgets we saw on the jQuery UI Themeroller page:



Title	Author	Original Language	First Published	Approximate Sales
A Brief History of Time	Stephen Hawking	English	1988	10 million
A Message to Garcia	Elbert Hubbard	English	1899	40 million
A Tale of Two Cities	Charles Dickens	English	1859	200 million
And Then There Were None	Agatha Christie	English	1939	100 million
Angels and Demons	Dan Brown	English	2000	39 million
Anne of Green Gables	Lucy Maud Montgomery	English	1908	50 million
Black Beauty: His Grooms and Companions: The autobiography of a horse	Anna Sewell	English	1877	50 million
Catch-22	Joseph Heller	English	1961	10 million
Charlie and the Chocolate Factory	Roald Dahl	English	1964	13 million
Charlotte's Web	E.B. White; illustrated by Garth Williams	English	1952	45 million

Showing 1 to 10 of 106 entries

However, you'll notice that the color scheme for the table rows is still lavender.

9. Let's make some adjustments to the color scheme. Open `demo_table_jui.css`. There are just a couple of lines to update. First, we'll find line 281 where the colors for the zebra-striping for the table are defined and update those to the colors we'd like to use as follows:

```
tr.odd {
  background-color: #f1f7fb;
}

tr.even {
  background-color: white;
}
```

   I'm going with a pale blue for odd rows and white for even rows to match the Cupertino style I selected earlier. Feel free to choose colors that match your own chosen theme.

10. Next, we'll change the color scheme for the currently sorted row. You'll find the CSS for the sorted odd rows at line 380. I'm going to change mine to a medium blue as follows:

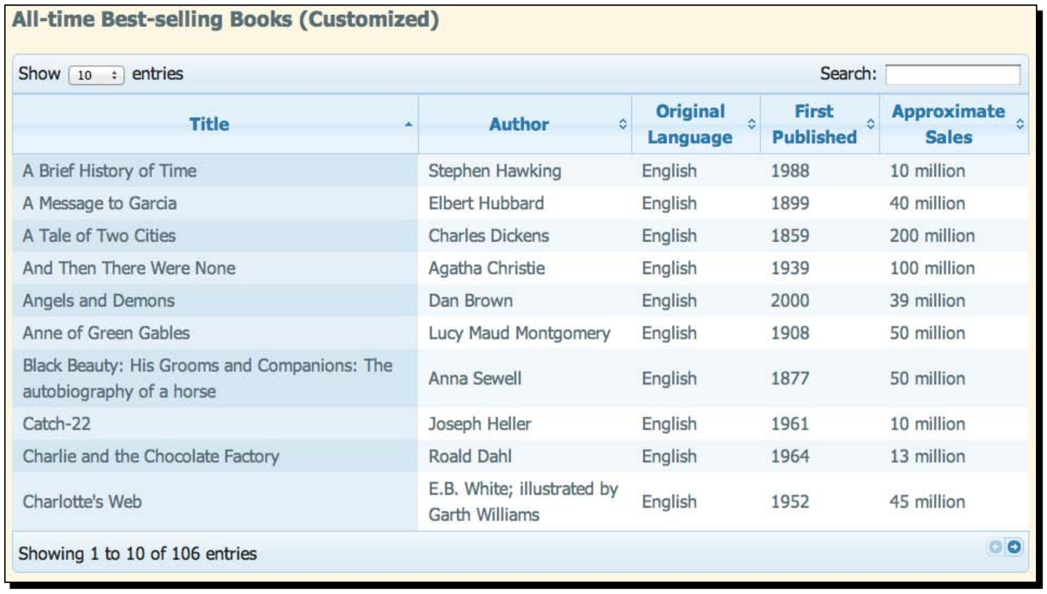
```
tr.odd td.sorting_1 {
  background-color: #d6e7f4;
}
```

11. And finally, we can find the CSS for the sorted even row at line 392. I'm going to change this to a light blue.

```
tr.even td.sorting_1 {
  background-color: #e4eff8;
}
```

   You can select your own colors that coordinate with your own chosen theme.

   Now, if you refresh the page in the browser, you'll see that the zebra-striping pattern of the table fits with our Themeroller theme.



All-time Best-selling Books (Customized)

Show 10 entries Search:

Title	Author	Original Language	First Published	Approximate Sales
A Brief History of Time	Stephen Hawking	English	1988	10 million
A Message to Garcia	Elbert Hubbard	English	1899	40 million
A Tale of Two Cities	Charles Dickens	English	1859	200 million
And Then There Were None	Agatha Christie	English	1939	100 million
Angels and Demons	Dan Brown	English	2000	39 million
Anne of Green Gables	Lucy Maud Montgomery	English	1908	50 million
Black Beauty: His Grooms and Companions: The autobiography of a horse	Anna Sewell	English	1877	50 million
Catch-22	Joseph Heller	English	1961	10 million
Charlie and the Chocolate Factory	Roald Dahl	English	1964	13 million
Charlotte's Web	E.B. White; illustrated by Garth Williams	English	1952	45 million

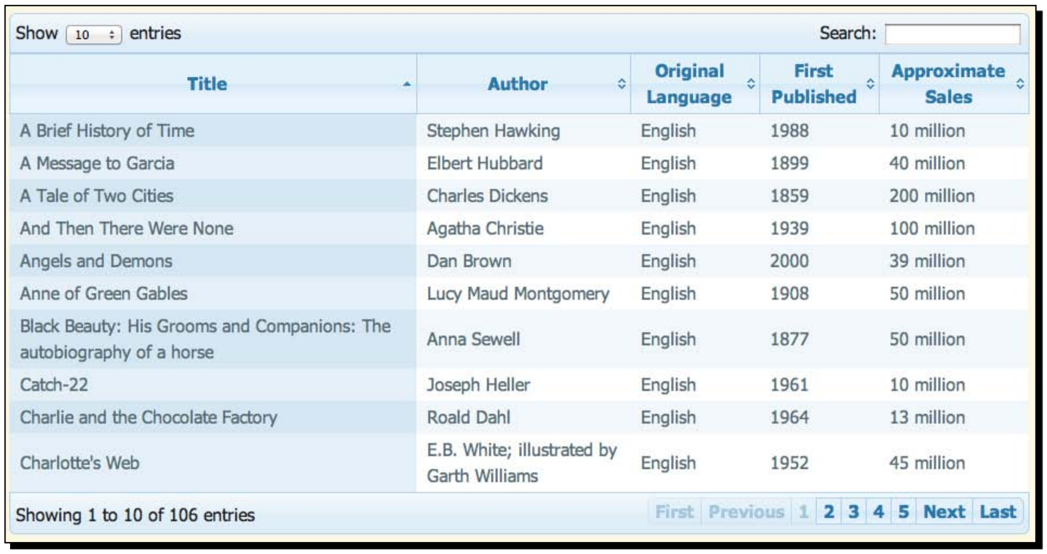
Showing 1 to 10 of 106 entries

12. Next up, we'll take a look at making a couple of other customizations to the data grid. First, let's change those simple next and previous pagination buttons into numbers. We'll pass another key/value pair to the `dataTable` method to replace the buttons with pagination numbers as follows:

```
$(document).ready(function(){
  $('#book-grid').dataTable({
    'sPaginationType': 'full_numbers',
    'bJQueryUI': true
  });
});
```

Remember to separate each key/value pair with a comma, but not to place a comma after the last key/value pair.

If you refresh the page in the browser, you'll see that the simple buttons have been replaced by pagination numbers as shown in the following screenshot:



Show 10 entries Search:

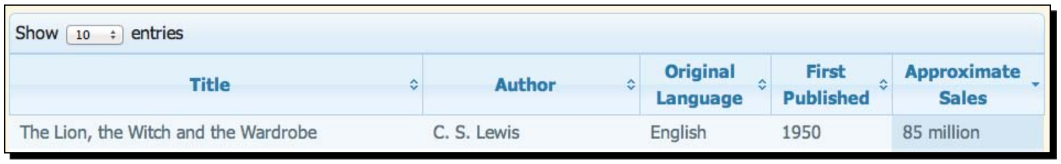
Title	Author	Original Language	First Published	Approximate Sales
A Brief History of Time	Stephen Hawking	English	1988	10 million
A Message to Garcia	Elbert Hubbard	English	1899	40 million
A Tale of Two Cities	Charles Dickens	English	1859	200 million
And Then There Were None	Agatha Christie	English	1939	100 million
Angels and Demons	Dan Brown	English	2000	39 million
Anne of Green Gables	Lucy Maud Montgomery	English	1908	50 million
Black Beauty: His Grooms and Companions: The autobiography of a horse	Anna Sewell	English	1877	50 million
Catch-22	Joseph Heller	English	1961	10 million
Charlie and the Chocolate Factory	Roald Dahl	English	1964	13 million
Charlotte's Web	E.B. White; illustrated by Garth Williams	English	1952	45 million

Showing 1 to 10 of 106 entries

13. We might decide that for this particular data table, the search function doesn't make sense. The DataTables plugin offers a way for us to disable individual features. To disable the search box filtering, we'll pass another key/value pair as follows:

```
$(document).ready(function(){
  $('#book-grid').dataTable({
    'sPaginationType': 'full_numbers',
    'bJQueryUI': true,
    'bFilter': false
  });
});
```

Refresh the page in the browser and you'll see that the search box is gone.



Show 10 entries

Title	Author	Original Language	First Published	Approximate Sales
The Lion, the Witch and the Wardrobe	C. S. Lewis	English	1950	85 million

14. You've probably noticed that by default, the DataTables plugin is sorting our table by the first column in ascending order, from A to Z. That might be fine in some cases, but in this case since we're listing the all-time best-selling books, we probably want to sort the table to show the books with the highest sales first. We'll pass in a new key/value pair to specify which column should be used for the default sort and which direction the sort should go.

```
$(document).ready(function(){
  $('#book-grid').dataTable({
    'sPaginationType': 'full_numbers',
    'bJQueryUI': true,
    'bFilter': false,
    'aaSorting': [[4, 'desc']]
  });
});
```

The key we're using is called `'aaSorting'`, and the value is the column number and sort direction inside two sets of square brackets. Don't forget that JavaScript starts counting at 0, not 1. So the fifth column in our table is actually column 4. Then, we want the highest number at the top, so we pass `'desc'` for descending order.

Refresh the page in the browser and you'll see that the books are now in order from highest sales to lowest sales. Note also that this default sort order doesn't affect your site visitor's ability to sort the table by any of the other columns in any order they'd like. The site visitor can still interact with your table. We're just re-defining the default view in a way that makes the most sense for the data we're presenting.

What just happened?

We took our basic data grid and took it a step further by customizing the appearance and behavior of the plugin. We learned how to use the jQuery UI Themeroller to create a custom theme for our data grid. Then we learned how to replace the simple pagination buttons with page numbers, disable searching the table, and how to set a default sort for the data grid.

Summary

In this chapter, we learned how to turn an ordinary HTML table into an interactive data grid. Our site visitors can now take advantage of sorting different columns of the table to view the data in different ways. Site visitors with JavaScript disabled simply see an ordinary HTML table that contains all of the data. Data grids aren't terribly exciting, but they can make dealing with large amounts of data worlds easier for your site visitors. Next up, we'll learn how to make forms both prettier and easier to use.

12
Improving Forms

If you've ever tried to work with web forms, you know what a headache they can be. Luckily, the authors of HTML5 are working hard to make sure that experience improves. We're all waiting patiently for browsers to support those nice new features, but in the meantime we have to build sites and turn out beautiful functioning forms.

In this chapter, you'll learn the following topics:

- Marking up a form with some of the new HTML5 attributes
- Placing the cursor in the first form field
- Using placeholder text in form fields
- Validating your site visitors' form entries
- Styling stubborn form elements such as file uploads and select drop downs

An HTML5 web form

We'll get started by taking advantage of some of the new attributes made available to us in HTML5. The great thing about these additions is that they are completely backward compatible. Browsers that don't know how to handle them will either ignore them or default to a simple text input, and our site visitors on older browsers will be able to use our forms without even knowing what they're missing.

First, a word of warning about web forms. A web form doesn't work by itself—it needs to have some fancy backend programming on a server somewhere to collect the form entries and process them, whether that means writing fields to the database or sending the form information via e-mail. Because of this, the forms we build in this chapter won't actually work—nothing will happen after clicking the **Submit** button on the form.

If you want to add a functioning web form to a project, you have a few options. They are as follows:

- You can learn to do server-side programming to handle your form, but server-side programming is well beyond the scope of this book.
- You can use a CMS that will likely include form handling either in its core functionality or as an add-on. Good candidates include Drupal, WordPress, and Joomla!.
- You can hire a server-side developer to get your form working. Or make friends with one and barter your design skills for their coding skills.
- You can use a web form service to handle all the server-side processing of your form. My personal favorite is WuFoo, which I have used for years without a single hiccup. (`http://wufoo.com`)

Any of these methods will help you create a working form to be included in your project. However, let's take a look at how we can make the front end of our form the best it can be.

Time for action – setting up an HTML5 web form

1. We'll get started with a simple HTML document and the associated files and folders, just like we set up in *Chapter 1*, *Designer, Meet jQuery*. We want to make sure to use the HTML5 doctype in our document type declaration at the top of the document:

   ```
   <!DOCTYPE html>
   ```

   After all those long and convoluted document type declarations used by HTML 4 and xHTML, this one is a breath of fresh air, isn't it?

2. Now, inside the `<body>` tag, open up a `<form>` tag as follows:

   ```
   <form action="#" id="account-form">
   </form>
   ```

   The `form` tag needs an `action` attribute in order to work. Since our forms are just dummy forms for scripting and styling purposes, we'll just use `#` as the value of that attribute. The value of the `action` attribute is usually a URL—the place on the server where we're going to send our form data for processing. We also added an `id` attribute to make it easy to select the form for CSS and JavaScript purposes later.

3. Next up, we'll create a section for our site visitor to create a username and password. We'll wrap these two fields up in a `fieldset` with a `legend` to group them together.

   ```
   <form action="#" id="account-form">
     <fieldset>
   ```

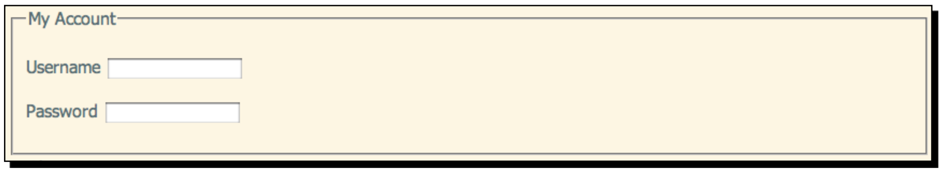
```
    <legend>My Account</legend>
    <p>
      <label for="username">Username</label>
      <input type="text" name="username" id="username"/>
    </p>
    <p>
      <label for="password">Password</label>
      <input type="password" name="password" id="password"/>
    </p>
  </fieldset>
</form>
```

I've wrapped each field and its associated label in a paragraph tag (`<p>`). There is a world of opinion out there on the best tags to use to mark up your form fields. Some developers swear by simple `<div>` tags, others like to make the form a list (`<ul>`) with each field a list item (`<li>`). Others like to use a definition list (`<dl>`) and place the labels inside the `<dt>` tags and the form fields inside the `<dd>` tag. At the end of the day, any of these will do just fine and your form will work as expected for your site visitors. Use whatever tags are your personal preference.

Look carefully at the HTML markup we've written so far for our form. There are a few important things to note. They are as follows:

- Each `<input>` has a `type` that is relevant to its purpose. **Username** has a `text` type and **Password** has a `password` type.
- Each `<input>` has a unique `id`. Remember that an `id` has to be unique on the page, so select the `id` of your form inputs carefully.
- Each `<input>` has a `name` attribute. This is passed to whatever code is handling your form on the server side. It's a common practice to use the same value for the `name` and `id` of a form element, but it's not compulsory. You may easily select a different value for the `id` anytime you'd like, but if you'd like to change the `name` value, you should first check with your server-side developer to make sure the code he or she has written will continue to work.
- Each `<label>` has a `for` attribute that associates it with a particular form element. The value in the `for` attribute is equal to the `id` of the form element with which it is associated (not the `name`). This makes some nice functionality available to our site visitors—clicking on a `label` will bring focus to the associated form element. This behavior is especially useful for checkbox and radio button inputs, which are small and can be difficult to click.

Each browser will have its own way of styling form elements, but here's what the **My Account** section looks like for me (Google Chrome on Mac OSX):

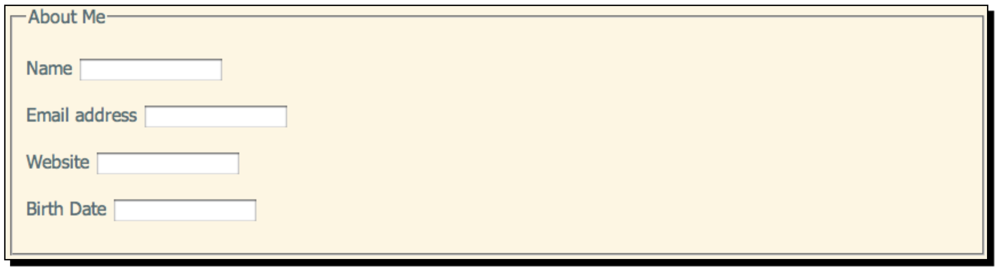


4. Next up, we'll create an **About Me** section for our form.

```
<fieldset>
  <legend>About Me</legend>
  <p>
    <label for="name">Name</label>
    <input type="text" name="name" id="name"/>
  </p>
  <p>
    <label for="email">Email address</label>
    <input type="email" name="email" id="email"/>
  </p>
  <p>
    <label for="website">Website</label>
    <input type="url" name="website" id="website"/>
  </p>
  <p>
    <label for="birthdate">Birth Date</label>
    <input type="date" name="birthdate" id="birthdate"/>
  </p>
</fieldset>
```

Again, the `text` type was used for the **Name** input, since names are strings. However, take a look at the `type` attribute for the **Email**, **Website**, and **Birth Date** fields. We're using the new HTML5 input types here. In browsers where these input types are not supported, these fields will look and work just like inputs with a `type` of `text`. But in browsers where these input types are recognized, they'll behave in a slightly different way. User input will automatically be validated by the browser. For example, if a site visitor types an invalid e-mail address into an input with the type `email`, the browser will warn them that they've entered an invalid e-mail address.

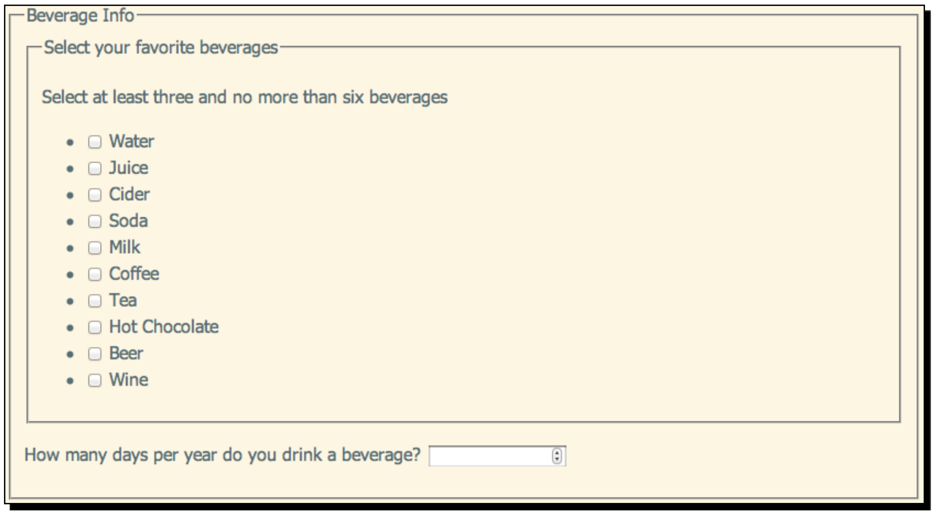
Also, on devices with soft keyboards, the keyboard keys will be altered to reflect the characters necessary for entering that data type. For example, an input with a type of `email` will open a keyboard with the `.` and the `@` showing on an iPhone or an iPad, making it easier for your site visitors on these devices to complete the required information.



5. The next section in my form will be a section about beverage preferences. I want the site visitor to select their favorite beverages from a list and then answer a question about how many days per year they drink a beverage. Here's a sample of what my list looks like:

```
<fieldset>
  <legend>Beverage Info</legend>
  <fieldset>
    <legend>Select your favorite beverages</legend>
    <p>Select at least three and no more than six beverages</p>
    <ul>
      <li>
        <input type="checkbox" name="favorites[]" id="bev-water" 
value="bev-water"/>
        <label for="bev-water">Water</label>
      </li>
      <li>
        <input type="checkbox" name="favorites[]" id="bev-juice" 
value="bev-juice"/>
        <label for="bev-juice">Juice</label>
      </li>
    </ul>
  </fieldset>
  <p>
    <label for="days">How many days per year do you drink a 
beverage?</label>
```

```
        <input type="number" name="days" id="days"/>
      </p>
    </fieldset>
```


Beverage Info
Select your favorite beverages
How many days per year do you drink a beverage?

Select at least three and no more than six beverages

- ☐ Water
- ☐ Juice
- ☐ Cider
- ☐ Soda
- ☐ Milk
- ☐ Coffee
- ☐ Tea
- ☐ Hot Chocolate
- ☐ Beer
- ☐ Wine

A few new things to note with the HTML we've used to mark up this section are as follows:

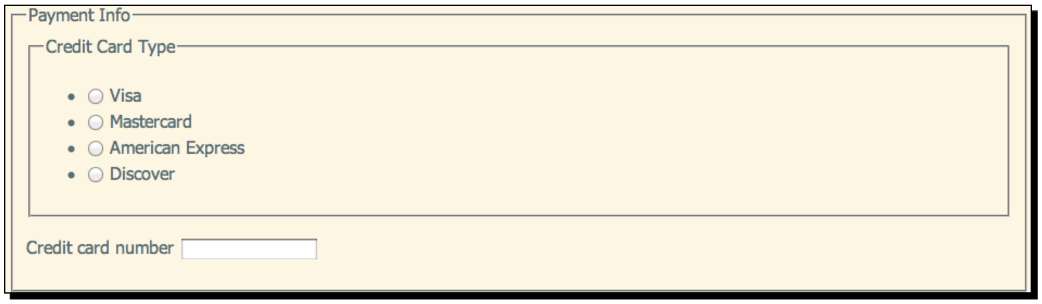
- `Fieldsets` can be nested. A `fieldset` is an excellent way to group a set of checkboxes or radio buttons together, and we can use the `legend` of the `fieldset` to create a header for our radio or checkbox group.
- A set of checkboxes are identified as such because they will all share the same `name`. As a site visitor can select more than one item in a set of checkboxes, we add square brackets (`[]`) to the end of the name so that the server will collect all of the answers into an array.
- Each checkbox in the set has its own unique `id` and `value`. The `id` and `value` do not necessarily have to match, but it's often easy to make them the same.
- Finally, the number of days per year is given an input type of `number`, since only a number would be acceptable here. Be careful with this input type. It is very strict and will not accept any non-numeric characters. Some bits of data appear to be numbers, but are actually strings—for example, telephone numbers and credit card numbers. If you wouldn't do some sort of math with your number then it shouldn't be the `number` input type.

6. The next section we'll add to our form is a payment information section:

```
<fieldset>
  <legend>Payment Info</legend>
  <fieldset>
    <legend>Credit Card Type</legend>
    <ul>
      <li>
        <input type="radio" name="cc-type" id="cc-visa" value="cc-
visa"/>
        <label for="cc-visa">Visa</label>
      </li>
      <li>
        <input type="radio" name="cc-type" id="cc-mastercard"
value="cc-mastercard"/>
        <label for="cc-mastercard">Mastercard</label>
      </li>
      <li>
        <input type="radio" name="cc-type" id="cc-amex" value="cc-
amex"/>
        <label for="cc-amex">American Express</label>
      </li>
      <li>
        <input type="radio" name="cc-type" id="cc-discover"
value="cc-discover"/>
        <label for="cc-discover">Discover</label>
      </li>
    </ul>
  </fieldset>
  <p>
    <label for="cc-number">Credit card number</label>
    <input type="text" name="cc-number" id="cc-number"/>
  </p>
</fieldset>
```

Much like the checkboxes, we've grouped a set of radio controls inside a `fieldset` with the `legend` acting as the header for this section. Just like checkboxes, a set of radio controls all share the same `name`, but each has its own unique `id` and `value`. However, in the case of radio buttons, only one can be selected at a time, so there is no need to mark them as an array.

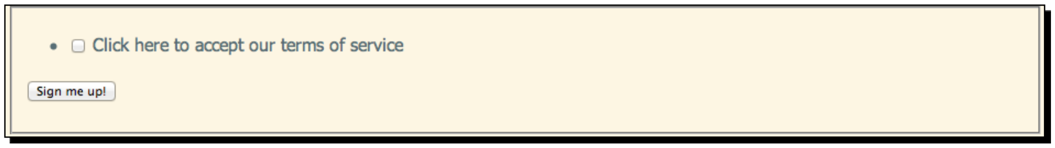
We've also added a field for collecting our site visitor's credit card number. Note that we've assigned an input type of `text` to this field. Even though a credit card number appears to be a number, we want to store it just as it is, and won't ever be adding to or subtracting from this number. Also, customers may wish to type spaces or hyphens in their credit card number.



7. Finally, we'll add a checkbox for our site visitor to accept our terms of service and a submit button for them to submit the form information to us.

```
<fieldset>
  <ul>
    <li>
      <input type="checkbox" name="tos" id="tos" value="tos"/>
      <label for="tos">Click here to accept our terms of service</
label>
    </li>
  </ul>
  <p>
    <input type="submit" value="Sign me up!"/>
  </p>
</fieldset>
```

The only new thing here is the **Submit** button. By default, an input with a type of `submit` will read **Submit**. We can change that by adding a `value` attribute with the text we'd like to actually appear on the button.

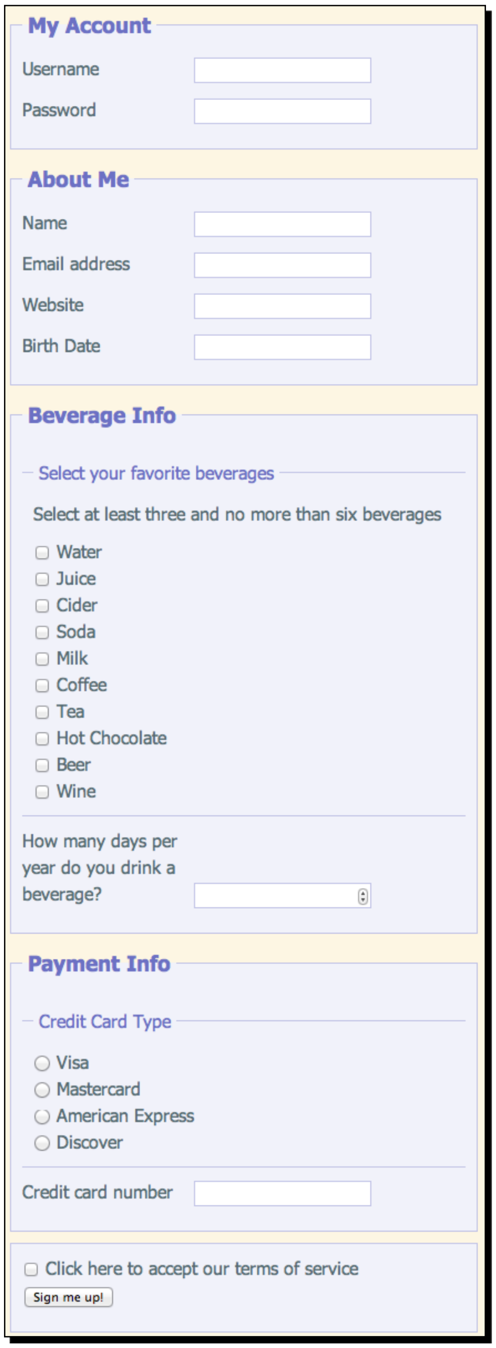


8. The only thing left to do is to style our form with a bit of CSS. The following is the CSS I've used for my simple form:

```
fieldset    { width:400px;margin:0;padding:10px;border:1px solid
#c1c3e6;background:#f1f2fa;margin-top:10px; }
fieldset fieldset  { border:0 none;border-top:1px solid
#c1c3e5;border-bottom:1px solid #c1c3e5;width:380px;margin-
bottom:10px; }
legend      { padding:3px 5px;color:#6c71c4;font-weight:bold;font-
size:1.2em; }
fieldset fieldset legend  { font-size:1em;font-weight:normal; }
fieldset p  { margin: 0 0 10px 0; }
fieldset ul { margin:0;padding:0;list-style:none; }
label { display:inline-block;width:150px; }
ul label  { display:inline;width:auto; }
input[type="text"],
input[type="password"],
input[type="email"],
input[type="url"],
input[type="date"],
input[type="number"]  { width:150px;border:1px solid
#c1c3e6;padding:4px; }
```

Note that the `type` attribute of our inputs can be used to select them for styling. In this case, I've styled them all identically, but it would also be possible to give each one its own set of styles if desired.

Here's how the form looks with my CSS. Feel free to get creative and write your own styles for the form.



My Account

Username	
Password	

About Me

Name	
Email address	
Website	
Birth Date	

Beverage Info

Select your favorite beverages

Select at least three and no more than six beverages

- ☐ Water
- ☐ Juice
- ☐ Cider
- ☐ Soda
- ☐ Milk
- ☐ Coffee
- ☐ Tea
- ☐ Hot Chocolate
- ☐ Beer
- ☐ Wine

How many days per year do you drink a beverage?

Payment Info

Credit Card Type

- ○ Visa
- ○ Mastercard
- ○ American Express
- ○ Discover

Credit card number

☐ Click here to accept our terms of service

Sign me up!

What just happened?

We took a look at some of the new HTML5 input types and how to use them properly to put together a web form. We saw how to use fieldsets and legends to group fields together under a heading and how to associate labels with form elements. We learned the proper use of the text, password, e-mail, URL, date, checkbox, radio, and number input types.

Setting focus

If you head over to `http://google.com`, you'll see that they've made it really easy for you to conduct a web search—as soon as the page is loaded up in the browser, the cursor is blinking in the search field. There are other sites on the web that behave this way too, making it quick and easy to get started filling in a form.

Any time you have a page where the site visitor's main task on the page will be completing a form, you can make things easy on your site visitor by placing the cursor into the first form field so they can just start typing. And it's wicked easy with jQuery. Here's how to do it.

Time for action – setting focus to the first field

We'll keep working with the sample form we set up in the last example. Here's how to set the focus to the first field in the form.

1. Open up your empty `scripts.js` file and add a document ready statement.

```
$(document).ready(function(){
  //code goes here
});
```

2. Next up, we want to select the first field in our form. There are many different ways to go about that. In this case, I'm going to use the `id` of the first form element.

```
$(document).ready(function(){
  $('#username');
});
```

3. All that's left to do is call the `focus()` method for that element.

```
$(document).ready(function(){
  $('#username').focus();
});
```

Now if you refresh the page in the browser, you'll see that the cursor is blinking in the **Username** field of the form—the very first field.

What just happened?

We used a couple of lines of jQuery to move focus to the first field in our form to make it easy for our site visitors to jump right into completing our form. It was as simple as selecting the first form element and then calling the `focus()` method for that element.

Placeholder text

Isn't it nice when you visit a site, and there's some soft grayed-out text in a form field giving you a hint about what you're supposed to put there? There are umpteen different jQuery plugins that have been written over the past several years to handle this because it can be a bit of a hassle.

However, I'm here with good news. HTML5 provides a `placeholder` attribute that can be used to create this kind of text in form fields automatically without any help from JavaScript. Of course, as with any other cutting-edge technology, browser support can be a bit lacking. We don't have the luxury of waiting years for browser support for this new feature to be universal—we have to build functioning websites now. You could continue using all those old jQuery plugins, but why not take advantage of support for the placeholder attribute if it's there and only use jQuery to fill in the gaps for those browsers that don't recognize it yet?

This type of script is called a **polyfill**. It's used to fill in functionality that might be missing from some browsers. If a browser does support the `placeholder` attribute, the polyfill script does nothing, and just lets the browser handle the placeholders. For all those site visitors without support for the `placeholder` attribute, the script springs into action, providing the placeholder text functionality for everyone.

Time for action – adding placeholder text

Follow these steps to add placeholder text to your form fields for as many of your site visitors as possible, whether or not their browser supports the new HTML5 placeholder attribute.

1. We'll keep using the same form that we've built in the last two sections. The first thing we'll do is revisit each form field and add a placeholder attribute where it makes sense. Here are some examples from my form:

```
<p>
  <label for="username">Username</label>
  <input type="text" name="username" id="username" placeholder="At
least 5 characters long"/>
</p>
```

Here, I've added a hint about the required length of the username.

```
<p>
  <label for="password">Password</label>
  <input type="password" name="password" id="password"
class="required" placeholder="Choose a secure password"/>
</p>
```

Because you can never say it too much, here I've reminded my site visitor to create a secure password.

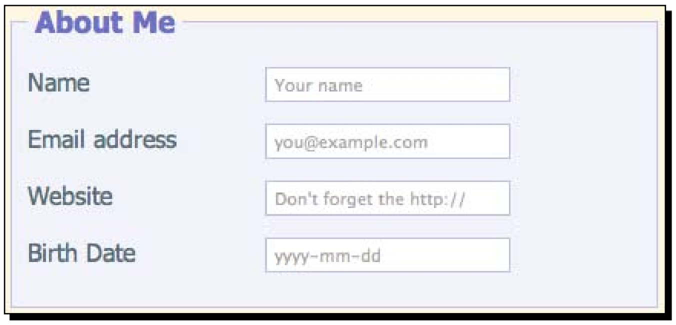
```
<p>
  <label for="website">Website</label>
  <input type="url" name="website" id="website" placeholder="Don't
forget the http://"/>
</p>
```

It can be helpful to remind site visitors that valid URLs include the protocol at the beginning.

```
<p>
  <label for="birthdate">Birth Date</label>
  <input type="date" name="birthdate" id="birthdate"
placeholder="yyyy-mm-dd"/>
</p>
```

Anytime a field requires special formatting, placeholder text can give a hint to the site visitor what that should be.

When you're finished adding placeholder text, view your page in Safari or Chrome to see the placeholder text in action.

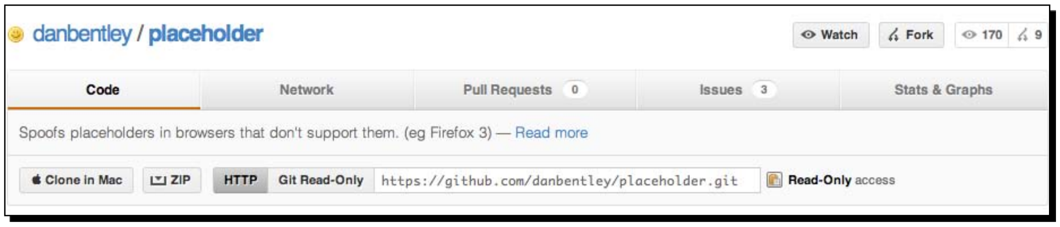


About Me

Name	Your name
Email address	you@example.com
Website	Don't forget the http://
Birth Date	yyyy-mm-dd

Now we need to add support for those browsers that don't yet support placeholder text.

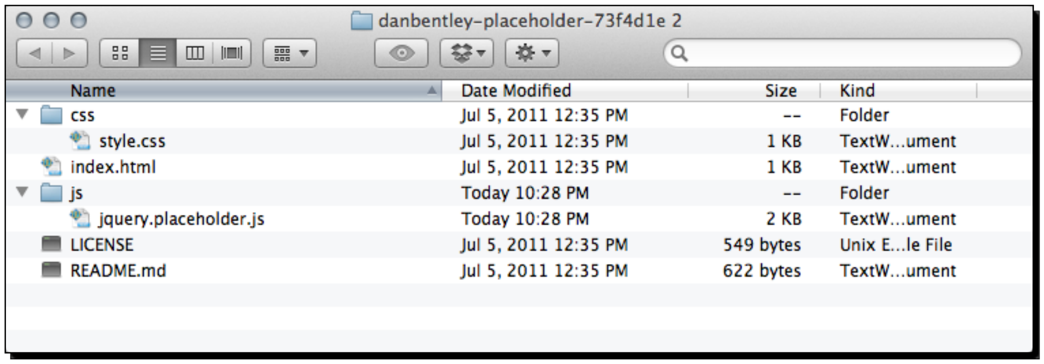
2. We'll use Dan Bentley's Placeholder polyfill. To download it, just head over to `https://github.com/danbentley/placeholder`. Just like the other plugins we've downloaded from GitHub, click on the **ZIP** button to download a zipped folder.



Spoofs placeholders in browsers that don't support them. (eg Firefox 3) — Read more

https://github.com/danbentley/placeholder.git

3. Unzip the folder and take a look inside. It's a pretty simple and straightforward plugin.



Name	Date Modified	Size	Kind
css	Jul 5, 2011 12:35 PM	--	Folder
style.css	Jul 5, 2011 12:35 PM	1 KB	TextW...ument
index.html	Jul 5, 2011 12:35 PM	1 KB	TextW...ument
js	Today 10:28 PM	--	Folder
jquery.placeholder.js	Today 10:28 PM	2 KB	TextW...ument
LICENSE	Jul 5, 2011 12:35 PM	549 bytes	Unix E...le File
README.md	Jul 5, 2011 12:35 PM	622 bytes	TextW...ument

You've got a sample `index.html` file, a `style.css` file, and a `jquery.placeholder.js` file, along with a license and a readme.

4. The good news about this plugin is that it works its magic just by being on the page. Copy `jquery.placeholder.js` to your own `scripts` folder. Then head down to the bottom of your page and attach the script to the page after jQuery and before your own `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.placeholder.js"></script>
<script src="scripts/scripts.js"></script>
```

Now, if you open the page in a browser that doesn't support placeholder attributes, you'll see placeholders working. Those browsers are Firefox 3.6 and lower, Safari 3 and lower, Internet Explorer 9 and lower, and Opera 10 and lower.

What just happened?

We used Dan Bentley's Placeholder polyfill to add placeholder support to browsers where it is lacking. We added `placeholder` attributes to the form fields where appropriate, then included Dan's script on our page to get those placeholder attributes working in as many browsers as possible.

Validating user entry

Sometimes it can feel frustrating for a site visitor when they have to submit a form several times over, correcting errors that they've made filling it out. Without JavaScript, the only way to validate the information the site visitor has entered is to wait for them to submit the form, then identify the issues on the server, and send back a page that contains the form along with any error messages that might help the site visitor correct the problem.

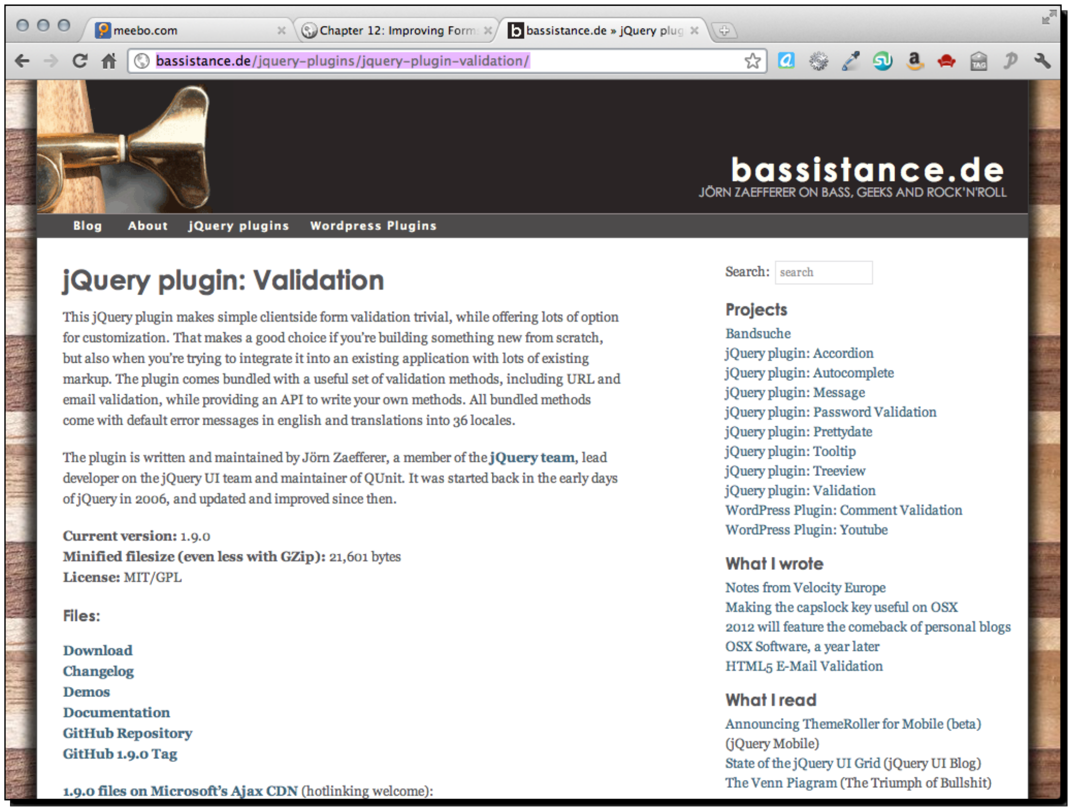
Showing errors as soon as they happen goes a long way toward making your form feel snappy and responsive and helping your site visitors submit the form correctly on the first try. In this section, we'll learn how to use the Validation plugin from Jörn Zaefferer. This plugin is powerful and flexible and can handle validation in several different ways. We'll take a look at the most straightforward way of adding client-side validation to your form.

Time for action – validating form values on the fly

We'll continue working with the form we've been creating through the last three sections. Follow these steps to validate user entry into the form:

1. The first thing we'll do is download the Validation plugin and get it attached to our page.

Head over to `http://bassistance.de/jquery-plugins/jquery-plugin-validation/` and click on the **Download** button in the **Files** section to download a ZIP file.



meebo.com
Chapter 12: Improving Form
bassistance.de » jQuery plug
bassistance.de/jquery-plugins/jquery-plugin-validation/
bassistance.de
JÖRN ZAEFFERER ON BASS, GEEKS AND ROCK'N'ROLL
Blog
About
jQuery plugins
Wordpress Plugins

jQuery plugin: Validation

This jQuery plugin makes simple clientside form validation trivial, while offering lots of option for customization. That makes a good choice if you're building something new from scratch, but also when you're trying to integrate it into an existing application with lots of existing markup. The plugin comes bundled with a useful set of validation methods, including URL and email validation, while providing an API to write your own methods. All bundled methods come with default error messages in english and translations into 36 locales.

The plugin is written and maintained by Jörn Zaefferer, a member of the **jQuery team**, lead developer on the jQuery UI team and maintainer of QUnit. It was started back in the early days of jQuery in 2006, and updated and improved since then.

Current version: 1.9.0
Minified filesize (even less with GZip): 21,601 bytes
License: MIT/GPL

Files:

Download
Changelog
Demos
Documentation
GitHub Repository
GitHub 1.9.0 Tag

1.9.0 files on Microsoft's Ajax CDN (hotlinking welcome):

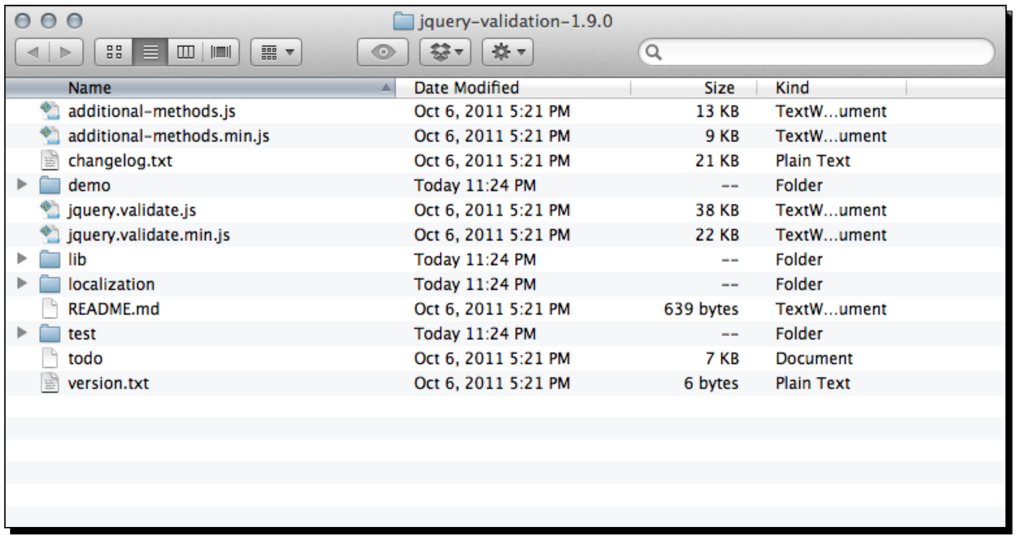
Search: search

Projects
Bandsuche
jQuery plugin: Accordion
jQuery plugin: Autocomplete
jQuery plugin: Message
jQuery plugin: Password Validation
jQuery plugin: Prettydate
jQuery plugin: Tooltip
jQuery plugin: Treeview
jQuery plugin: Validation
WordPress Plugin: Comment Validation
WordPress Plugin: Youtube

What I wrote
Notes from Velocity Europe
Making the capslock key useful on OSX
2012 will feature the comeback of personal blogs
OSX Software, a year later
HTML5 E-Mail Validation

What I read
Announcing ThemeRoller for Mobile (beta) (jQuery Mobile)
State of the jQuery UI Grid (jQuery UI Blog)
The Venn Piagram (The Triumph of Bullshit)

2. Open up the ZIP file and take a look at what we've got.



Name	Date Modified	Size	Kind
additional-methods.js	Oct 6, 2011 5:21 PM	13 KB	TextW...ument
additional-methods.min.js	Oct 6, 2011 5:21 PM	9 KB	TextW...ument
changelog.txt	Oct 6, 2011 5:21 PM	21 KB	Plain Text
demo	Today 11:24 PM	--	Folder
jquery.validate.js	Oct 6, 2011 5:21 PM	38 KB	TextW...ument
jquery.validate.min.js	Oct 6, 2011 5:21 PM	22 KB	TextW...ument
lib	Today 11:24 PM	--	Folder
localization	Today 11:24 PM	--	Folder
README.md	Oct 6, 2011 5:21 PM	639 bytes	TextW...ument
test	Today 11:24 PM	--	Folder
todo	Oct 6, 2011 5:21 PM	7 KB	Document
version.txt	Oct 6, 2011 5:21 PM	6 bytes	Plain Text

There's a lot going on here. Several different JavaScript files, a changelog, and so on. Remember how I said this plugin is powerful and can handle lots of different approaches to validation? That's what all this is for. Handling form validation in just about any old crazy situation you might find yourself in.

Luckily, though, our situation is pretty simple, so we don't have to do anything complicated.

3. Copy `jquery.validate.min.js` to your own `scripts` folder and attach it to your page.

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.placeholder.js"></script>
<script src="scripts/jquery.validate.min.js"></script>
```

In this case, there's no dependency between the placeholder script and the validation script, so it doesn't matter which order they appear in, as long as they're both after jQuery itself.

4. Next, we're going to go back through our form and add some information that the Validation plugin will use. Let's start with the username field:

```
<p>
  <label for="username">Username</label>
  <input type="text" name="username" id="username" placeholder="At
least 5 characters long" minlength="5" maxlength="20"
class="required"/>
</p>
```

This is a required field—any site visitor who completes this form must select a username, so I'll simply add a `class` of `required`. I can use that class name to create a special style for this form field with CSS if I'd like. Even if I don't, Validation will use this to make sure this field is filled in.

Next, all usernames must be between 5 and 20 characters long. So I've added a `minlength` and `maxlength` attribute.

5. Next up is the password field, which is also required. So I'll add the required class.

```
<p>
  <label for="password">Password</label>
  <input type="password" name="password" id="password"
class="required" placeholder="Choose a secure password"/>
</p>
```

While I'm at it, I'll add the required class to the e-mail field too.

```
<p>
  <label for="email">Email address</label>
  <input type="email" name="email" id="email" placeholder="you@
example.com" class="required"/>
</p>
```

6. Next, let's take a look at that list of favorite beverages. Remember we had a note on there for the site visitor to select at least three but no more than six? We can actually enforce that with the Validation plugin. Go to the first checkbox in the series and add `minlength` and `maxlength` attributes as follows:

```
<li>
  <input type="checkbox" name="favorites[]" id="bev-water"
value="bev-water" maxlength="6" minlength="3"/>
  <label for="bev-water">Water</label>
</li>
```

We only have to add this on the first checkbox, not all of them. Validation is smart enough to figure out that we're talking about this set of checkboxes.

7. Now, let's take a look at the field where we ask the site visitor how many days per year they drink a beverage. Obviously, since there are only 365 days in a year, that's the highest number they could enter in this field. So we'll add a `max` attribute to specify the highest possible number.

```
<p>
  <label for="days">How many days per year do you drink a
beverage?</label>
  <input type="number" name="days" id="days" max="365"/>
</p>
```

8. And that brings us to the payment section. Whatever we're selling, it's not free, so we're going to require both the credit card type and credit card number. To require entry for radio buttons, we just have to add the `required` class to the first radio button in the set.

```
<li>
  <input type="radio" name="cc-type" id="cc-visa" value="cc-visa"
class="required"/>
  <label for="cc-visa">Visa</label>
</li>
```

We don't have to make any other changes to the radio button series.

9. Now, let's handle the credit card number itself. We need to add the `required` class. We also need to add a `creditcard` class to validate that the number entered is, in fact, a valid credit card number.

```
<p>
  <label for="cc-number">Credit card number</label>
  <input type="text" name="cc-number" id="cc-number"
placeholder="xxxxxxxxxxxxxxxx" class="creditcard required"/>
</p>
```

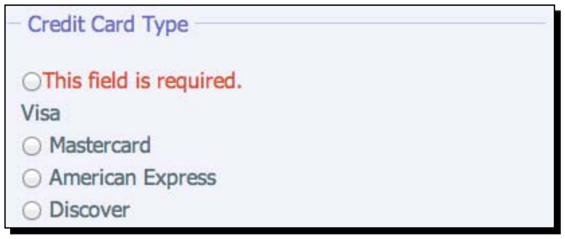
10. And at the bottom of our form, we have our **Terms of Service** checkbox. This is required too, so we'll add the `required` class.

```
<li>
  <input type="checkbox" name="tos" id="tos" class="required"
value="tos"/>
  <label for="tos">Click here to accept our terms of service</
label>
</li>
```

11. Now, we just need to call the `validate()` method that Validation makes available to us. Inside your document ready statement, select the form and call the `validate()` method.

```
$(document).ready(function(){
  $('#username').focus();
  $('#account-form').validate();
});
```

12. Now if you refresh the page in the browser, you'll see that you can't submit the form without filling anything in—the required fields will be marked with an error message saying the field is required. If you try to type an invalid URL or e-mail address into the **Website** or **Email address** fields, you'll get an error message letting you know there's a problem to be corrected. Just one problem—those error messages are sort of in a weird place for our checkboxes and radio buttons.



That doesn't really help people understand exactly what's going on. Luckily, Validation allows us to add our own error messages to the page wherever we'd like them to display.

13. We're going to add an error message after the list of credit card type radio buttons.

```
  <li>
      <input type="radio" name="cc-type" id="cc-discover"
value="cc-discover"/>
      <label for="cc-discover">Discover</label>
    </li>
  </ul>
  <label for="cc-type" class="error">Select a credit card type!</
label>
</fieldset>
```

We'll add a `<label>`. The `for` attribute will refer to the `name` of the field—in this case, all the radio buttons share the `cc-type` name. We'll add a class of `error`, and add whatever error message we'd like inside.

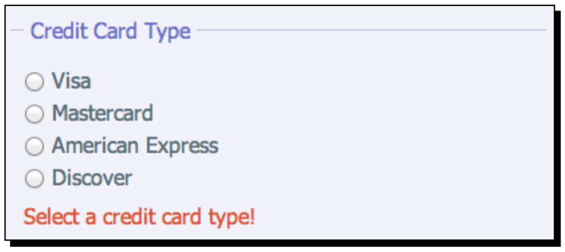
Note that for this case, the `for` attribute of our `label` is referring to the `name` of the field rather than the ID. This is a special case created by the Validation plugin. If you're not using custom error messages with the Validation plugin, then your label's `for` attribute should always reference the `id` of the form element.

14. Next, we don't want those error messages showing up on the page unless they're needed. We'd also like them to display in red so they stick out and are easy to find. Open your `styles.css` file and add some styles for the error message:

```
label.error { display:none;width:360px;color:#dc522f;margin-
top:5px; }
```

We're adding a width since I've set my other labels to be short and floated to the left. And we're adding a little margin for some space between the error message and the field it's referring to.

Now if you refresh the browser and try to submit the form without selecting a credit card type, you'll get the error message in a much better place as follows:



Select a credit card type!

15. Next we need to do the same thing for our favorite beverages and our **Terms of Service** checkbox: Here's what we'll add for favorite beverages:

```
    <li>
      <input type="checkbox" name="favorites[]" id="bev-wine"
value="bev-wine"/>
      <label for="bev-wine">Wine</label>
    </li>
  </ul>
  <label for="favorites[]" class="error">Please select at least
three and no more than six favorite beverages</label>
</fieldset>
```

And here's what we'll add for **Terms of Service**:

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

```
      <input type="checkbox" name="tos" id="tos"
class="required"/>
      <label for="tos">Click here to accept our terms of service</
label>
    </li>
  </ul>
  <label for="tos" class="error">You must accept our terms of
service</label>
  <p>
    <input type="submit"/>
  </p>
</fieldset>
```

Now, if you refresh the page in the browser and try to submit the form without completing required fields or try to enter invalid information in the form, you'll get an appropriate error message as soon as the problem is detected.

What just happened?

We used the Validation plugin to add some simple client-side validation to our form. The simplest way to use the Validation plugin is to simply add some class names and attributes to your form elements. Validation will take care of the rest—it's smart enough to recognize the HTML5 input types and validate those and offers some other useful validation rules such as required fields, a maximum number value, minimum and maximum lengths, and credit card numbers. We dropped in a line of CSS to style the error messages the way we wanted.

Improving appearance

If you've tried styling web forms with CSS, then you've probably discovered that some form elements, like text inputs and buttons, are pretty easy to style. There are a few quirks, but once you get those figured out, you can get those form elements looking just about any way you'd like. Other form elements, however, are much more stubborn and don't respond much, if at all, to CSS styles. It's so frustrating to design a lovely form only to realize that it's technically impossible.

These troublesome form elements are:

```
<select>
<input type="file">
<input type="checkbox">
<input type="radio">
```

Not only are these four form elements impossible to style with CSS, they also look radically different from one browser and operating system to another, leaving us with little control over the appearance of our form. Let's see how Pixel Matrix's Uniform plugin can help us out.

Time for action – improving form appearance

Follow these steps to take advantage of the styling options made possible by the Uniform plugin:

1. We'll get started with a basic HTML file and associated files and folders, just like we set up in *Chapter 1*, *Designer, Meet jQuery*. For this example, in the body of the HTML document, we're going to set up a simple form with examples of each type of hard-to-style form element. Get started with a `<form>` tag:

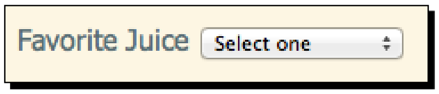
```
<form id="pretty-form" action="#">
</form>
```

2. Then, inside our form we'll add our form elements. We'll start off with a `select` drop down:

```
<fieldset>
  <legend>Select your favorite juice</legend>
  <p>
    <label for="juice">Favorite Juice</label>
    <select id="juice" name="juice">
      <option>Select one</option>
      <option value="orange">Orange Juice</option>
      <option value="grape">Grape Juice</option>
      <option value="grapefruit">Grapefruit Juice</option>
      <option value="cranberry">Cranberry Juice</option>
      <option value="tomato">Tomato Juice</option>
      <option value="pineapple">Pineapple Juice</option>
      <option value="apple">Apple Juice</option>
    </select>
  </p>
</fieldset>
```

We're following all the same rules we followed for the last form, making sure the form works properly and is accessible.

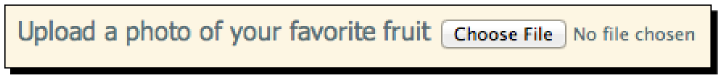
Exactly what this `<select>` looks like will depend on your browser and operating system, but here's how mine looks in Chrome on Mac OSX:



3. Next, we'll add a file input.

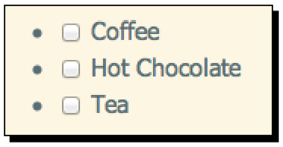
```
<fieldset>
  <legend>Fruit Picture</legend>
  <p>
    <label for="fruit-photo">Upload a photo of your favorite
fruit</label>
    <input type="file" id="fruit-photo" name="fruit-photo"/>
  </p>
</fieldset>
```

Hard to believe this innocent-looking little tag could be the source of so much styling headache, but there you are. Here's how it looks in Chrome on Mac OSX:



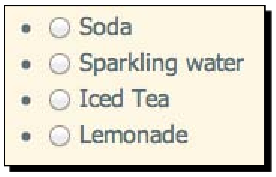
4. Next up, let's add a few checkboxes as follows:

```
<fieldset>
  <legend>Which hot beverages do you enjoy?</legend>
  <ul>
    <li>
      <input type="checkbox" name="hot-bevs[]" id="hot-coffee">
      <label for="hot-coffee">Coffee</label>
    </li>
    <li>
      <input type="checkbox" name="hot-bevs[]" id="hot-chocolate">
      <label for="hot-chocolate">Hot Chocolate</label>
    </li>
    <li>
      <input type="checkbox" name="hot-bevs[]" id="hot-tea">
      <label for="hot-tea">Tea</label>
    </li>
  </ul>
</fieldset>
```



5. And then some radio buttons.

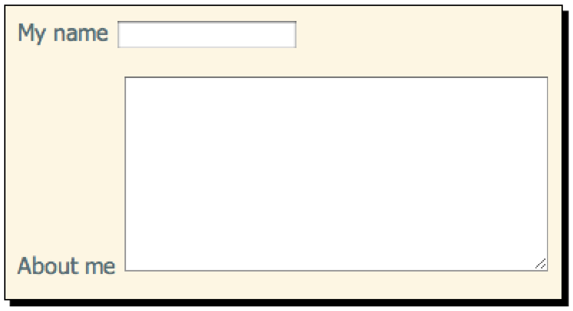
```
<fieldset>
  <legend>Select your favorite soft drink</legend>
  <ul>
    <li>
      <input type="radio" name="soft-drinks" id="soda"/>
      <label for="soda">Soda</label>
    </li>
    <li>
      <input type="radio" name="soft-drinks" id="sparkling-
water"/>
      <label for="sparkling-water">Sparkling water</label>
    </li>
    <li>
      <input type="radio" name="soft-drinks" id="iced-tea"/>
      <label for="iced-tea">Iced Tea</label>
    </li>
    <li>
      <input type="radio" name="soft-drinks" id="lemonade"/>
      <label for="lemonade">Lemonade</label>
    </li>
  </ul>
</fieldset>
```



6. And the last thing we'll add to our form is just a few easily styleable elements, so that we can learn how to style these to match our Uniform styles:

```
<fieldset>
  <legend>Some other stuff about me</legend>
  <p>
    <label for="name">My name</label>
    <input type="text" id="name" name="name"/>
  </p>
  <p>
```

```
        <label for="about-me">About me</label>
        <textarea rows="10" cols="40" id="about-me" name="about-me"></
textarea>
    </p>
</fieldset>
<p class="buttons">
    <input type="submit"/>
    <input type="reset"/>
</p>
```



What just happened?

Now we've got our unstyled form set up. Exactly what our form looks like will depend on your browser and operating system. We followed all the rules established earlier in this chapter for setting up a correct and accessible form. Except this time, we've included some difficult-to-style form elements. Let's take a look now at how we can use the Uniform plugin to get our form looking consistent across as many browsers as possible.

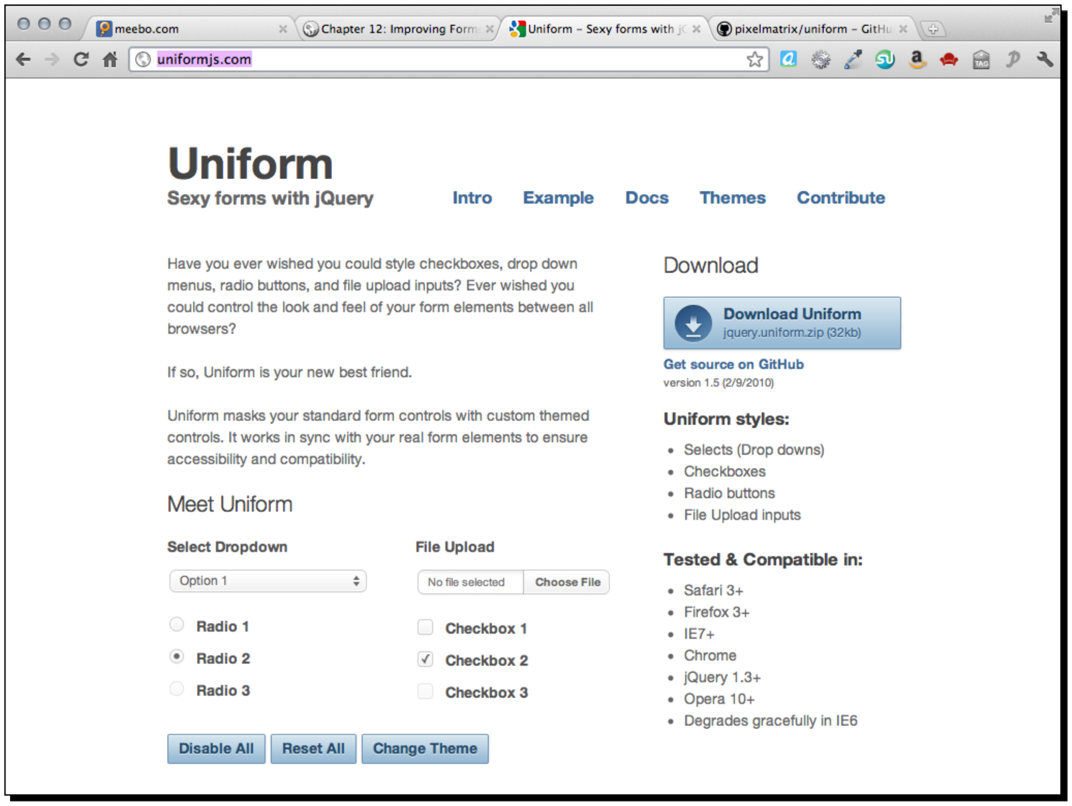
Styling the unstylable

If you want to take a little time out and try writing some CSS to style these form elements, you'll see that there's not much that touches them. Some of them don't seem to be affected by CSS at all, and when they are, it's not always in the way that you'd expect. No wonder these form fields give everyone so much trouble. JQuery to the rescue.

Time for action – adding uniform for styling the unstylable

Follow these steps to use the Uniform plugin to gain styling control over your form elements:

1. Let's get the Uniform plugin and take a look at how that works. Head over to `http://uniformjs.com/` and click on the big **Download Uniform** button.



meebo.com | Chapter 12: Improving Form | Uniform – Sexy forms with jQ | pixelmatrix/uniform – GitHu

uniformjs.com

Uniform
Sexy forms with jQuery

Intro Example Docs Themes Contribute

Have you ever wished you could style checkboxes, drop down menus, radio buttons, and file upload inputs? Ever wished you could control the look and feel of your form elements between all browsers?

If so, Uniform is your new best friend.

Uniform masks your standard form controls with custom themed controls. It works in sync with your real form elements to ensure accessibility and compatibility.

Meet Uniform

Select Dropdown: Option 1

File Upload: No file selected | Choose File

Radio 1
Radio 2 ●
Radio 3

Checkbox 1
Checkbox 2 ✓
Checkbox 3

Disable All | Reset All | Change Theme

Download

Download Uniform
jquery.uniform.zip (32kb)

Get source on GitHub
version 1.5 (2/9/2010)

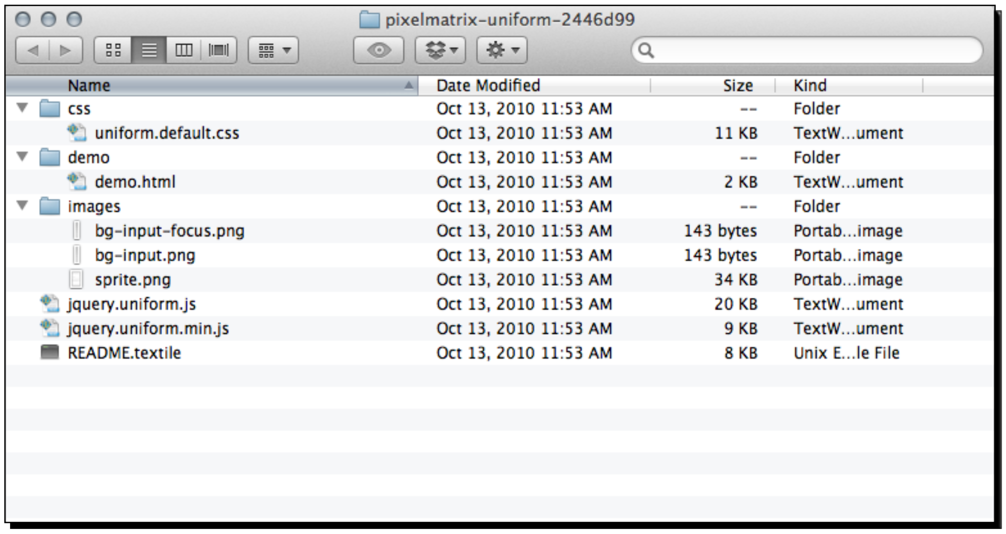
Uniform styles:

- Selects (Drop downs)
- Checkboxes
- Radio buttons
- File Upload inputs

Tested & Compatible in:

- Safari 3+
- Firefox 3+
- IE7+
- Chrome
- jQuery 1.3+
- Opera 10+
- Degrades gracefully in IE6

2. Unzip the folder and take a look inside.

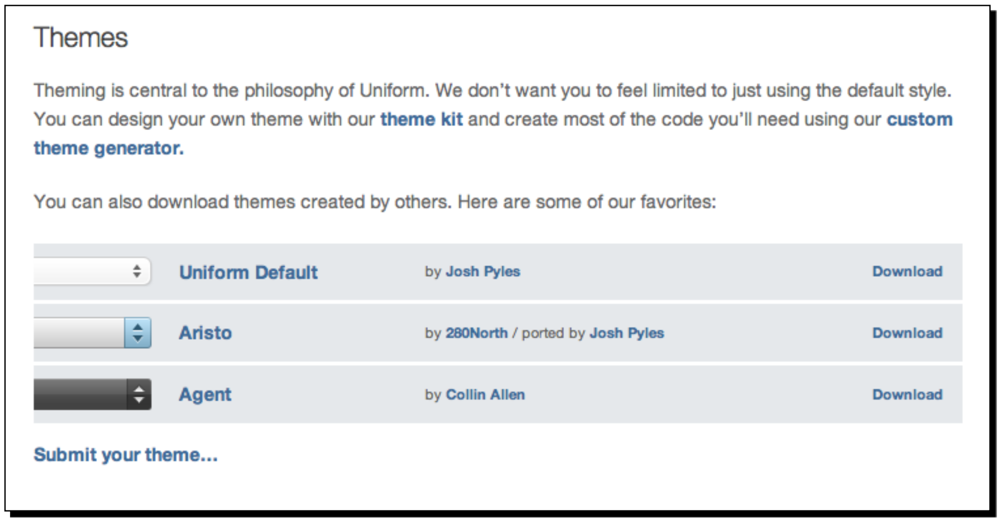


pixelmatrix-uniform-2446d99

Name	Date Modified	Size	Kind
css	Oct 13, 2010 11:53 AM	--	Folder
uniform.default.css	Oct 13, 2010 11:53 AM	11 KB	TextW...ument
demo	Oct 13, 2010 11:53 AM	--	Folder
demo.html	Oct 13, 2010 11:53 AM	2 KB	TextW...ument
images	Oct 13, 2010 11:53 AM	--	Folder
bg-input-focus.png	Oct 13, 2010 11:53 AM	143 bytes	Portab...image
bg-input.png	Oct 13, 2010 11:53 AM	143 bytes	Portab...image
sprite.png	Oct 13, 2010 11:53 AM	34 KB	Portab...image
jquery.uniform.js	Oct 13, 2010 11:53 AM	20 KB	TextW...ument
jquery.uniform.min.js	Oct 13, 2010 11:53 AM	9 KB	TextW...ument
README.textile	Oct 13, 2010 11:53 AM	8 KB	Unix E...le File

This is pretty straightforward, right? Some styles, a demo, some images, and two versions of the Uniform plugin—one minified and one not. We've seen this before.

By default, Uniform comes with a default stylesheet and images. However, other styles are available. Back on `uniformjs.com`, if you click on **Themes** in the navigation, you'll see the themes that are currently available. I really like the look of Aristo, so I'm going to download that.



Themes

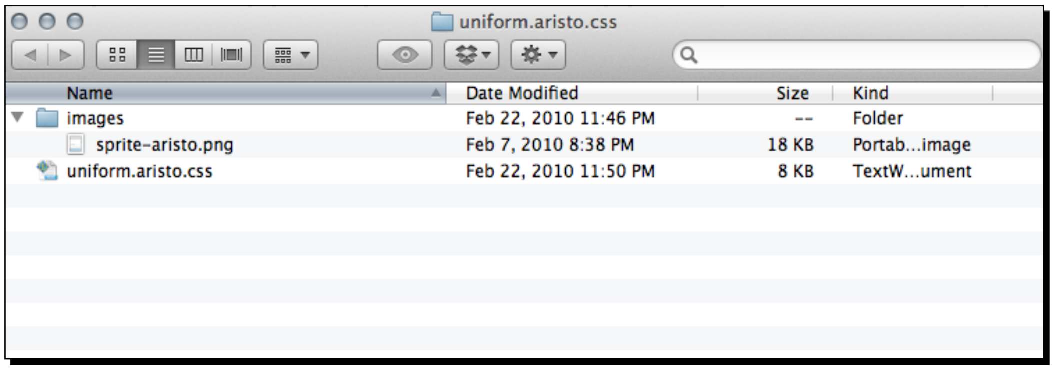
Theming is central to the philosophy of Uniform. We don't want you to feel limited to just using the default style. You can design your own theme with our **theme kit** and create most of the code you'll need using our **custom theme generator.**

You can also download themes created by others. Here are some of our favorites:

Uniform Default	by Josh Pyles	Download
Aristo	by 280North / ported by Josh Pyles	Download
Agent	by Collin Allen	Download

Submit your theme...

This gets me a simple ZIP file with just some css and images inside:



uniform.aristo.css

Name	Date Modified	Size	Kind
images	Feb 22, 2010 11:46 PM	--	Folder
sprite-aristo.png	Feb 7, 2010 8:38 PM	18 KB	Portab...image
uniform.aristo.css	Feb 22, 2010 11:50 PM	8 KB	TextW...ument

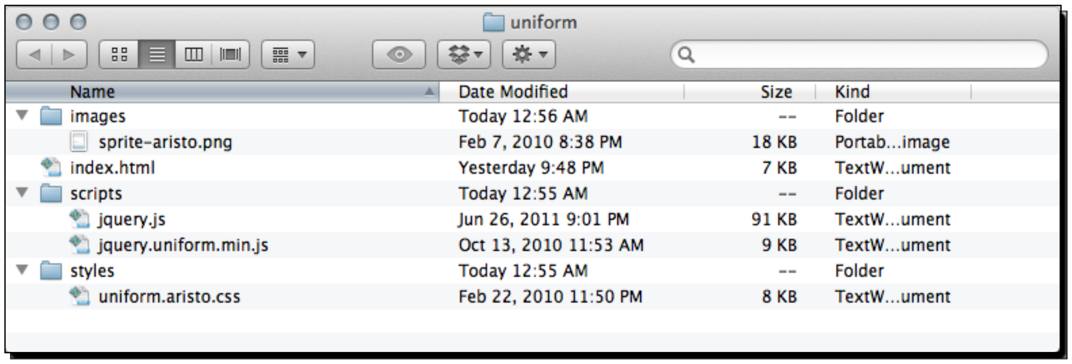
3. Next, we need to get these files into our own project and attached to our HTML page. Let's start with the JavaScript. Copy `jquery.uniform.min.js` to your own `scripts` folder and attach the Uniform script between jQuery and your own `scripts.js` file:

```
<script src="scripts/jquery.js"></script>
<script src="scripts/jquery.uniform.min.js"></script>
<script src="scripts/scripts.js"></script>
</body>
```

4. Now copy the CSS file for the theme you'd like to use to your own `styles` folder and attach it in the head of the document:

```
<head>
  <title>Chapter 12: Improving Forms</title>
  <link rel="stylesheet" href="styles/uniform.aristo.css"/>
  <link rel="stylesheet" href="styles/styles.css"/>
```

5. The last thing we need to grab is the associated images. Copy the contents of your chosen theme's images folder to your own `images` folder. Your own project's structure should now look similar to the following screenshot:



Name	Date Modified	Size	Kind
images	Today 12:56 AM	--	Folder
sprite-aristo.png	Feb 7, 2010 8:38 PM	18 KB	Portab...image
index.html	Yesterday 9:48 PM	7 KB	TextW...ument
scripts	Today 12:55 AM	--	Folder
jquery.js	Jun 26, 2011 9:01 PM	91 KB	TextW...ument
jquery.uniform.min.js	Oct 13, 2010 11:53 AM	9 KB	TextW...ument
styles	Today 12:55 AM	--	Folder
uniform.aristo.css	Feb 22, 2010 11:50 PM	8 KB	TextW...ument

6. Now, we're ready to call the `uniform()` method to style our unstylable form elements. Open up your `scripts.js` file, and insert a document ready statement:

```
$(document).ready(function(){
  //our code will go here
});
```

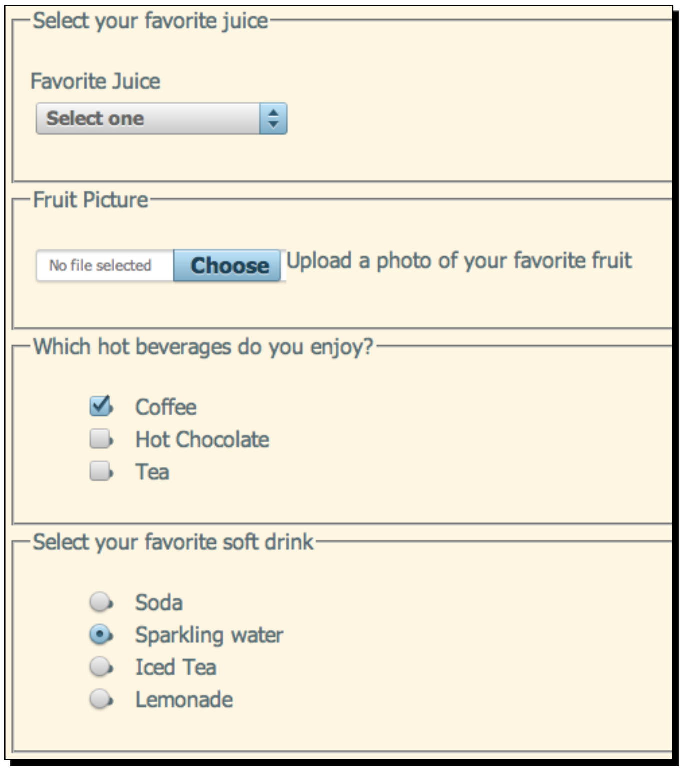
7. Uniform allows us to pick and choose which form elements we'd like to style. In this case, we want to style all four stubborn elements, so our selector will be:

```
$(document).ready(function(){
  $('select, input:checkbox, input:radio, input:file');
});
```

8. Then, all that's left to do is call the `uniform()` method:

```
$(document).ready(function(){
  $('select, input:checkbox, input:radio, input:file').uniform();
});
```

Now if you refresh the page in the browser, you'll see these stubborn and unstylable form elements now match the Uniform theme that you've selected.


Select your favorite juice

Favorite Juice

Select one

Fruit Picture

No file selected | Choose | Upload a photo of your favorite fruit

Which hot beverages do you enjoy?

- [✓] Coffee
- [ ] Hot Chocolate
- [ ] Tea

Select your favorite soft drink

- ( ) Soda
- (●) Sparkling water
- ( ) Iced Tea
- ( ) Lemonade

There are still some funky CSS things to take care of, and our fieldsets, legends, buttons, and text inputs don't match. Let's write a bit of CSS to bring it all together.

Styles for all

We still have some CSS things to clean up—our lists of checkboxes and radio buttons still have their bullets and our text inputs, buttons, fieldsets, and so on are still unstyled. Let's style everything to match the Uniform theme we've selected.

Time for action – styling the styleable

1. Open up your `styles.css` file. We'll start off by styling the fieldsets and legends:

```
fieldset  {
  background: #fff;
  border: 1px dotted #83b0ca;
  margin: 10px 20px 0 20px;
  padding:10px;
  }

legend  {
  background: #bed6e3;
  border:1px solid #8fb7cf;
  color: #1C4257;
  padding: 0 5px;
  box-shadow:2px 2px 2px rgba(0,0,0,0.2);
  }
```

I've selected shades of blue that match the Aristo theme that I selected. If you chose a different theme, feel free to use different colors and styles to match your chosen theme.

2. Next, we'll style some of the container elements we're using in the form:

```
fieldset p  {
  margin: 0 0 10px 0;
  }

fieldset ul  {
  list-style: none;
  margin: 0;
  padding: 0;
  }

label  {
  display: block;
  }

ul label  {
  display: inline;
  width: auto;
  }

p.buttons  {
  margin: 20px;
  }
```

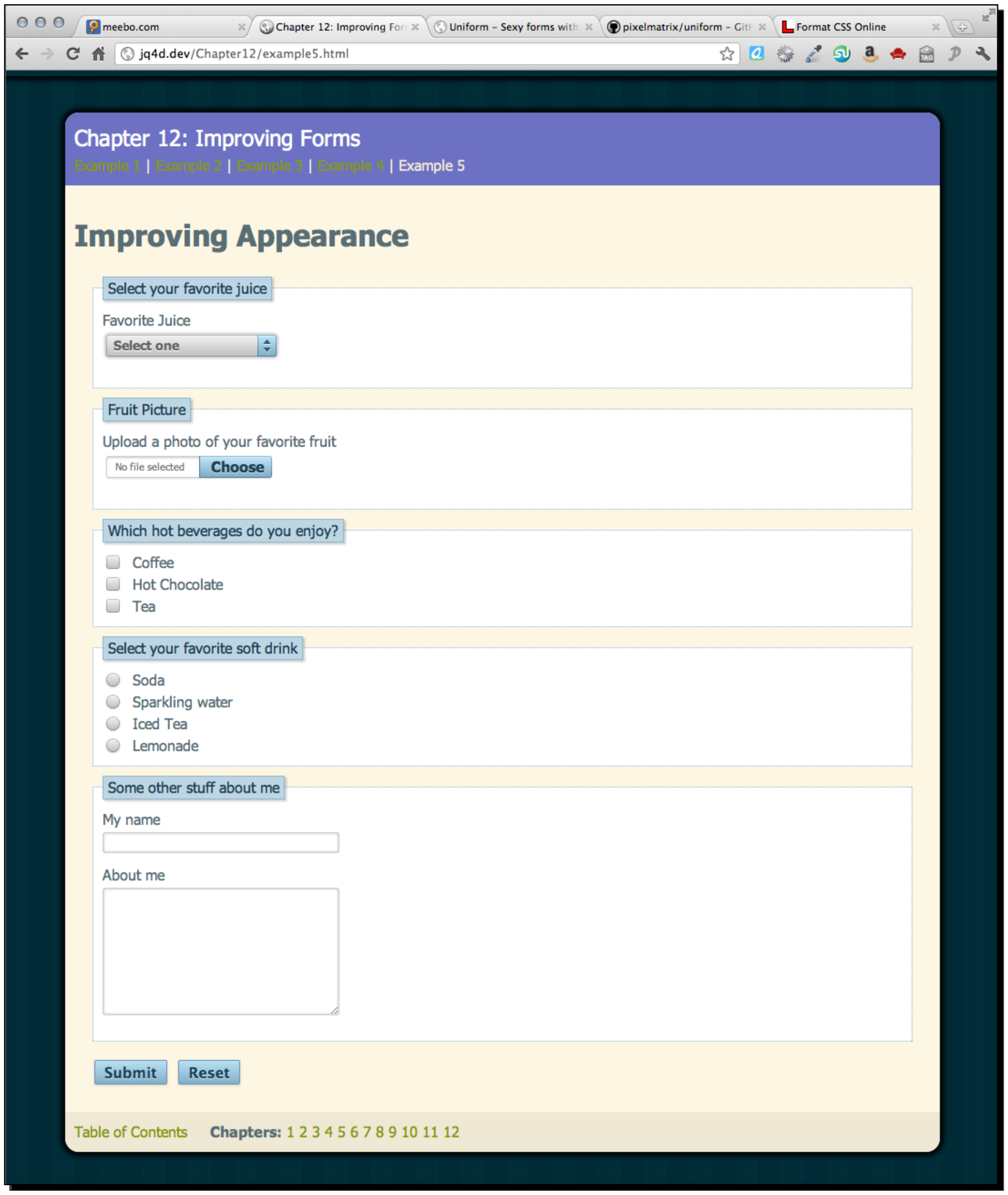
3. Next, we'll add some styles to that text input and textarea so they match our Aristo form elements:

```
input[type="text"],
textarea  {
  border: 1px solid #ccc;
  border-radius: 3px;
  box-shadow: inset 0 0 4px rgba(0,0,0,0.3);
  moz-border-radius: 3px;
  moz-box-shadow: inset 0 0 4px rgba(0,0,0,0.3);
  padding: 4px;
  webkit-border-radius: 3px;
  webkit-box-shadow: inset 0 0 4px rgba(0,0,0,0.3);
  width: 250px;
  }
```

4. And last, but not least, we'll style our buttons. The Aristo theme makes use of a nice blue gradient, so I'm going to use a gradient for my buttons. I'll have to write quite a lot of code for supporting all the browsers, but here it is:

```
input[type='submit'],
input[type='reset'] {
  background: rgb(185,224,245);
  background: linear-gradient(top, rgba(185,224,245,1)
0%,rgba(131,176,202,1) 100%);
  background: -moz-linear-gradient(top, rgba(185,224,245,1) 0%,
rgba(131,176,202,1) 100%);
  background: -ms-linear-gradient(top, rgba(185,224,245,1)
0%,rgba(131,176,202,1) 100%);
  background: -o-linear-gradient(top, rgba(185,224,245,1)
0%,rgba(131,176,202,1) 100%);
  background: -webkit-gradient(linear, left top, left bottom,
color-stop(0%,rgba(185,224,245,1)), color-stop(100%,rg
ba(131,176,202,1)));
  background: -webkit-linear-gradient(top, rgba(185,224,245,1)
0%,rgba(131,176,202,1) 100%);
  border: solid 1px #6e93b0;
  border-radius: 2px;
  box-shadow: rgba(0,0,0,0.15) 0px 1px 3px;
  color: #1C4257;
  cursor: pointer;
  display: inline-block;
  filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#b9e0f5', endColorstr='#83b0ca',GradientType=0 );
  filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#eef3f8', endColorstr='#96b9d4',GradientType=0 );
  font-size: 1em;
  font-weight: bold;
  height: 27px;
  line-height: 26px;
  margin-right: 5px;
  moz-border-radius: 2px;
  moz-box-shadow: rgba(0,0,0,0.15) 0px 1px 3px;
  padding: 0 10px;
  text-shadow: rgba(255,255,255,0.5) 0px 1px 0px;
  webkit-border-radius: 2px;
  webkit-box-shadow: rgba(0,0,0,0.15) 0px 1px 3px;
  }

input[type='submit']:hover,
input[type='reset']:hover {
  color: #0b1b24;
  }

input[type='submit']:active,
input[type='reset']:active  {
  background: rgb(131,176,202);
  background: linear-gradient(top, rgba(131,176,202,1)
0%,rgba(185,224,245,1) 100%);
  background: -moz-linear-gradient(top, rgba(131,176,202,1) 0%,
rgba(185,224,245,1) 100%);
  background: -ms-linear-gradient(top, rgba(131,176,202,1)
0%,rgba(185,224,245,1) 100%);
  background: -o-linear-gradient(top, rgba(131,176,202,1)
0%,rgba(185,224,245,1) 100%);
  background: -webkit-gradient(linear, left top, left bottom,
color-stop(0%,rgba(131,176,202,1)), color-stop(100%,rg
ba(185,224,245,1)));
  background: -webkit-linear-gradient(top, rgba(131,176,202,1)
0%,rgba(185,224,245,1) 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(
startColorstr='#83b0ca', endColorstr='#b9e0f5',GradientType=0 );
  }
```

I'm adding a subtle text color change on hover and reversing the gradient when the buttons are clicked. Now, refresh the page in the browser and take a look at our beautiful form.


meebo.com
Chapter 12: Improving For
Uniform – Sexy forms with
pixelmatrix/uniform – Git
Format CSS Online
jq4d.dev/Chapter12/example5.html

Chapter 12: Improving Forms

Example 1 | Example 2 | Example 3 | Example 4 | Example 5

Improving Appearance

Select your favorite juice

Favorite Juice

Select one

Fruit Picture

Upload a photo of your favorite fruit

No file selected Choose

Which hot beverages do you enjoy?

Coffee
Hot Chocolate
Tea

Select your favorite soft drink

Soda
Sparkling water
Iced Tea
Lemonade

Some other stuff about me

My name

About me

Submit Reset

Table of Contents Chapters: 1 2 3 4 5 6 7 8 9 10 11 12

What just happened?

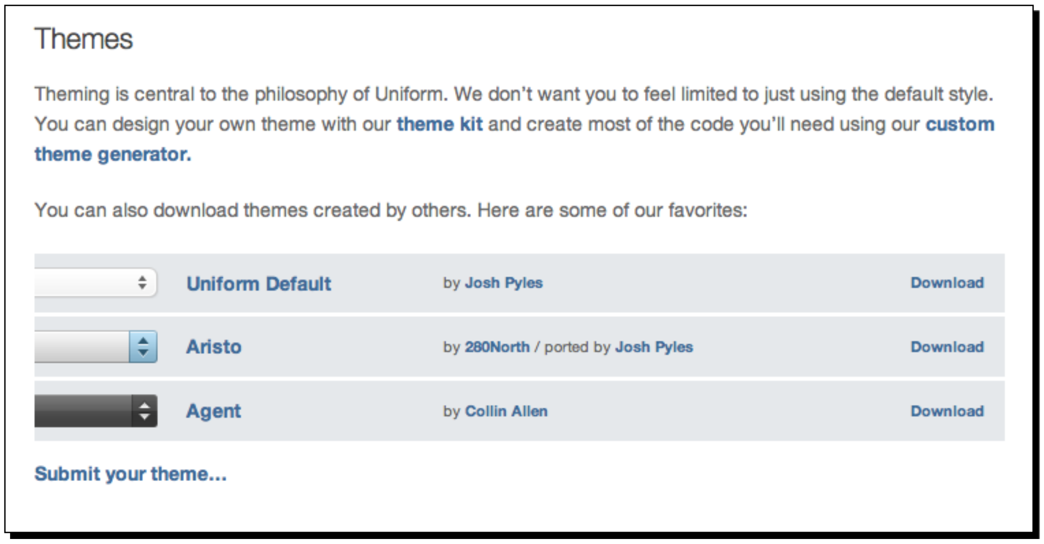
We used the Pixelmatrix's Uniform jQuery plugin to style formerly stubborn and unstyleable form elements. We chose one of the pre-made themes and attached all relevant CSS and images to our page, then selected each type of form element we wanted to style and called the `uniform()` method. We then used our CSS skills to style the other form elements, a simple text input, a textarea, and some buttons, to match the theme we selected. The result is a gorgeous form that will look consistent across different browsers and will still work perfectly for users with JavaScript disabled.

Our own theme

Sure, this Aristo theme is nice, but what if it doesn't match our site? Do we have any other option? Of course we do! If none of the prebuilt themes match your site, you can make your own theme using your own styles and colors to match any site you'd like. In fact, Pixelmatrix has made it super easy. Here's how you do it:

Time for action – creating a custom uniform theme

1. Start off by downloading the theme kit from Pixelmatrix. It's available in the themes section on `uniformjs.com`:



Themes

Theming is central to the philosophy of Uniform. We don't want you to feel limited to just using the default style. You can design your own theme with our **theme kit** and create most of the code you'll need using our **custom theme generator.**

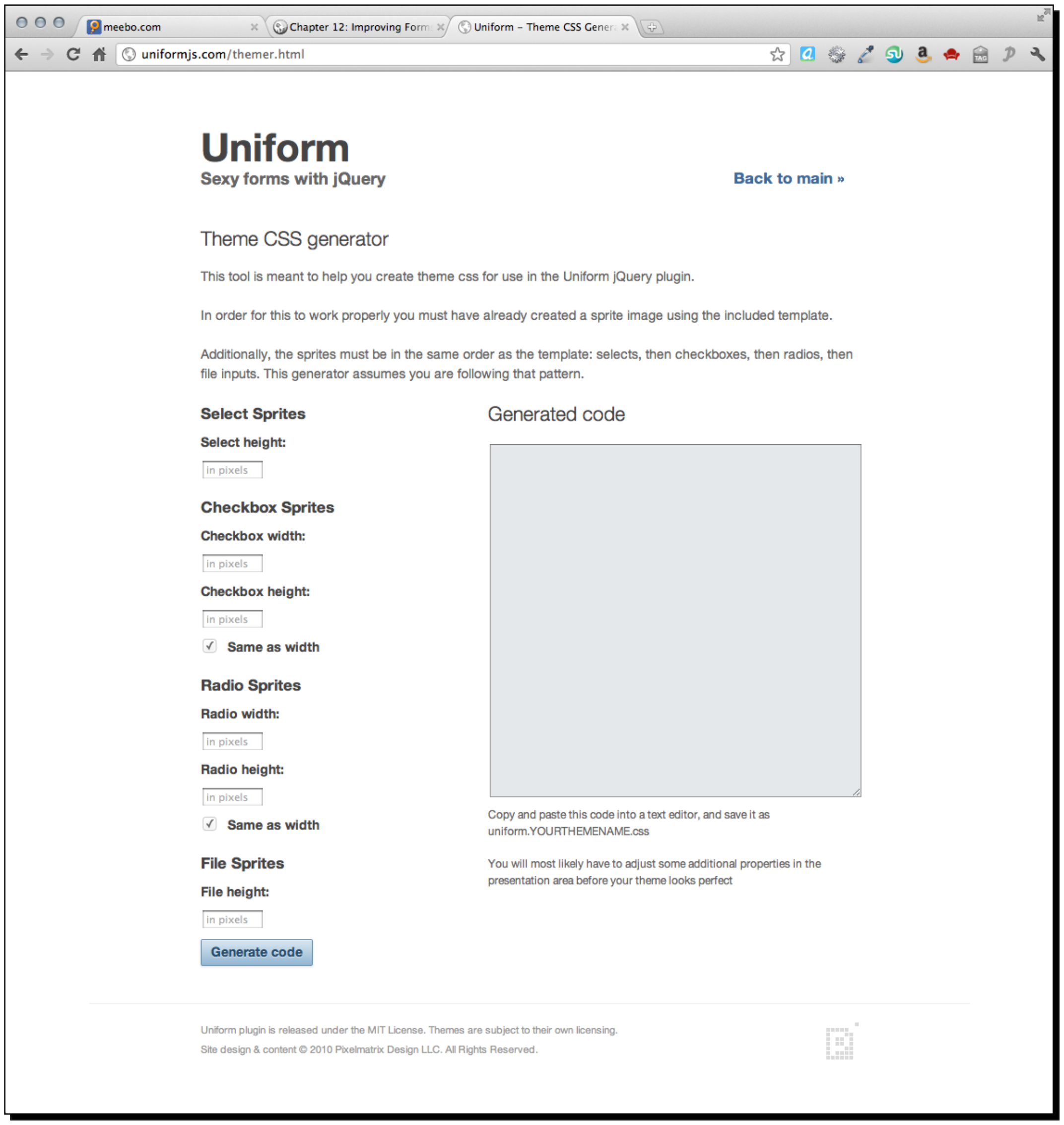
You can also download themes created by others. Here are some of our favorites:

	Uniform Default	by **Josh Pyles**	**Download**
	Aristo	by **280North** / ported by **Josh Pyles**	**Download**
	Agent	by **Collin Allen**	**Download**

Submit your theme...

2. Unzip the folder and inside you'll find two PSD files—`sprite.psd` and `sprites.psd`. Open up `sprite.psd` in Photoshop and style the form elements to your heart's content. You can change the sizes of the elements if you'd like to have larger or smaller form elements. `Sprites.psd` is only for explaining what each style is for. You can use it as a reference to make sure you get all the possibilities covered, but you won't actually need to use it to create your theme.

3. When your sprite is ready, head over to `http://uniformjs.com/themer.html`.


meebo.com
Chapter 12: Improving Form
Uniform – Theme CSS Gener
uniformjs.com/themer.html

Uniform

Sexy forms with jQuery

Back to main »

Theme CSS generator

This tool is meant to help you create theme css for use in the Uniform jQuery plugin.

In order for this to work properly you must have already created a sprite image using the included template.

Additionally, the sprites must be in the same order as the template: selects, then checkboxes, then radios, then file inputs. This generator assumes you are following that pattern.

Select Sprites

Select height: in pixels

Checkbox Sprites

Checkbox width: in pixels

Checkbox height: in pixels

✓ Same as width

Radio Sprites

Radio width: in pixels

Radio height: in pixels

✓ Same as width

File Sprites

File height: in pixels

Generate code

Generated code

Copy and paste this code into a text editor, and save it as uniform.YOURTHEMENAME.css

You will most likely have to adjust some additional properties in the presentation area before your theme looks perfect

Uniform plugin is released under the MIT License. Themes are subject to their own licensing.

Site design & content © 2010 Pixelmatrix Design LLC. All Rights Reserved.

Fill out the form with height of your select sprite, the width and height of your checkboxes and radio buttons, and the height of your file input. Then click **Generate code**. The CSS that you'll need to have Uniform work with your sprite will be generated for you. Copy and paste it into a CSS file and save it to your project.

4. Attach your new CSS file to your HTML document and save your sprite as a PNG file to the `images` folder in your project, and you should be all set. You might find a few things that need some minor tweaks, but setting up a custom Uniform theme is that straightforward.

> If you'd like to contribute your theme back to the Uniform community for other designers and developers to use, you can submit it to Pixelmatrix by e-mailing a zip of your theme to `josh@pixelmatrixdesign.com`.

What just happened?

We learned how to use the theme kit and custom theme CSS generator provided by Pixelmatrix to quickly and easily create our own Uniform theme.

Summary

Well, that wraps up the chapter on forms. We learned how to use the new HTML5 form elements properly to create a form that functions perfectly and is accessible to boot. We learned how to focus the first field in the form, use placeholder text in all browsers, validate our site visitor's form input and style those stubborn and notoriously unstyleable form elements. Now you've got an arsenal of tools on your side to create gorgeous-looking forms that enhance your site visitors' experience on your site. And best of all, they all degrade gracefully for users with JavaScript disabled since we approached our forms with the progressive enhancement mindset—first building out a working form, then layering in enhancements for those site visitors whose browsers support them.

I know that JavaScript can be a scary subject for designers. Kudos to you for sticking with me to the end of the book! I hope now that you have a basic understanding of jQuery and feel sure that you'll be able to tackle your next JavaScript challenge with confidence. You know how to put the jQuery library to good use to enhance your sites. You know how to find good plugins to make coding up interactions quick and easy. You know how CSS and JavaScript can work together to enhance the site visitor's experience on your site. And you know that there is no shortage of tutorials, resources, help forums, articles, and discussions online to help you along if you get stuck.

For its part, jQuery gets better with every release—sleeker, faster, and more capable. The jQuery team is careful to keep the documentation updated so you'll always be able to figure out just how to use each method. The jQuery team is smart and quick, and new jQuery updates are being announced on a regular schedule. All of this points to a lively and useful library that will only continue to grow in popularity across the Web. It's a favorite of many coders, from experienced hackers to beginners like you.

I hope that you've enjoyed this book and that it's given you many new ideas for interactive elements you can design and build for your sites. Be sure to stay connected to the jQuery community—it will be your best resource moving forward with further improving and growing your JavaScript skills.

Index